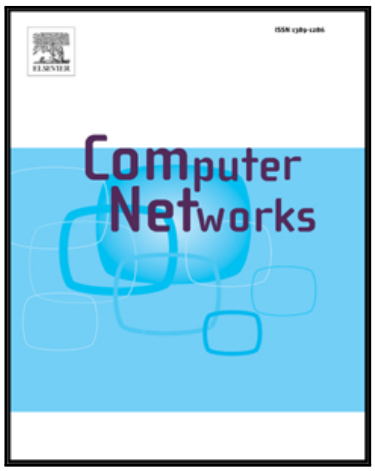


Methodology, Measurement and Analysis of Flow Table Update Characteristics in Hardware OpenFlow Switches

Maciej Kuźniar, Peter Perešíni, Dejan Kostić, Marco Canini

PII: S1389-1286(18)30081-1
DOI: 10.1016/j.comnet.2018.02.014
Reference: COMPNW 6412

To appear in: *Computer Networks*

Received date: 30 June 2017
Revised date: 25 December 2017
Accepted date: 14 February 2018

Please cite this article as: Maciej Kuźniar, Peter Perešíni, Dejan Kostić, Marco Canini, Methodology, Measurement and Analysis of Flow Table Update Characteristics in Hardware OpenFlow Switches, *Computer Networks* (2018), doi: 10.1016/j.comnet.2018.02.014

Methodology, Measurement and Analysis of Flow Table Update Characteristics in Hardware OpenFlow Switches

Maciej Kuźniar [☆], Peter Perešíni[◇☆], Dejan Kostić[†,☆☆], and Marco Canini[‡]

[*]Google, [◇]Una liated, [†]KTH Royal Institute of Technology, [‡]KAUST

Abstract

Software-Defined Networking (SDN) and OpenFlow are actively being standardized and deployed. These deployments rely on switches that come from various vendors and di□er in terms of performance and available features. Understanding these di□erences and performance characteristics is essential for ensuring successful and safe deployments.

We propose a systematic methodology for SDN switch performance analysis and devise a series of experiments based on this methodology. The methodology relies on sending a stream of rule updates, while relying on both observing the control plane view as reported by the switch and probing the data plane state to determine switch characteristics by comparing these views. We measure, report and explain the performance characteristics of flow table updates in six hardware OpenFlow switches. Our results describing rule update rates can help SDN designers make their controllers e□cient. Further, we also highlight di□erences between the OpenFlow specification and its implementations, that if ignored, pose a serious threat to network security and correctness.

Keywords: Software-Defined Networking, switch, flow table updates, measurements

1. Introduction

Software-Defined Networking (SDN), and OpenFlow in particular are increasingly being standardized and deployed by many including the hyperscale companies like Google, Microsoft, Facebook, etc. that consider SDN to be the future of computer networks [1, 2, 3, 4]. This means that the number of SDN developers creating exciting new frameworks [5, 6, 7] as well as network administrators that are using a variety of SDN controllers is rapidly growing.

In OpenFlow, the control plane involves a controller communicating with OpenFlow agents running (as pat of the firmware) on the switch-local control plane to instruct them how to configure the data plane by sending flow modification commands that place rules in the forwarding tables. A single deployment can use one or more type of OpenFlow switches, and the developer typically assumes that if the switch conforms to a specification, it will perform as a well-behaved black box. SDN's transition from research to production means that real deployments are demanding new levels of reliability and performance requirements that are necessary for production environments. For example, consistent network update schemes [8, 9, 10] are trying to ensure that packets do not get lost while new forwarding rules are being installed. Schemes also exist for ensuring congestion-free updates [11] and for scheduling rule installations to minimize rule installation time [9, 12, 13, 14]. All of these assume quick rule installation latency, and many rely on update confirmations from the switch-local control plane before proceeding to the next step.

[☆]Work done at EPFL, Switzerland

[☆☆]Corresponding author: Dejan Kostic, address: Kistagangen 16, 16440 Kista, Sweden, email: dmk@kth.se

Section	Key finding
4.1	Barriers should not be trusted! Updates are often applied in hardware hundreds of milliseconds after a barrier that confirms them. One of the tested switches reorders updates despite the barriers.
4.2	In the worst case, rules were installed minutes after a switch confirmed the installation.
4.3	Firmware is often responsible for switch faulty behavior and low performance.
4.4	Rule modification operation is non-atomic and switch may even flood packets for a transient period of time!
4.5	Rule updates get reordered even if there is a barrier between them and they affect the same flows. Some switches ignore priorities.
5.1	Few outstanding requests are enough to saturate the switch.
5.2	Rule updates get slower as the flow table occupation gets higher.
5.3	Using rule priorities may degrade update performance by orders of magnitude. Rule update patterns matter and switches can take advantage of an update locality.
5.4	Barriers are costly at some switches.

Table 1: Summary of key findings presented in this paper.

Initially, sporadic OpenFlow switch performance measurements were reported [15, 16, 17]. A structured set of measurements was reported in the pioneering work on OFLOPS [18], a generic framework for OpenFlow switch evaluation. For example, it was shown that there are issues with the implementation of the barrier command and it is important to understand and optimize SDN control in the presence of switch diversity [19]. This article extends our previous reports on switch performance [20] with new findings, deeper explanations and measurement results for double the number of switches (six instead of three), which include both low- and high-end devices.

While measuring switch performance might appear to be a simple task, it nevertheless has its own challenges. The biggest issue is that each switch under test has a lot of "quirks" that result in unexplained performance deviations from its usual behavior. Therefore, thoroughly evaluating and explaining these phenomena takes a substantial effort. For example, finding the absolute rule installation count or rate that takes the switch across the performance threshold can require a large number of experiments. Moreover, there is a very large number of combinations of rule modification commands to test with.

In this paper, we set out to advance the general understanding of OpenFlow switch performance. Specifically, the focus of this paper is on analyzing control plane performance and flow table update rate in hardware OpenFlow switches that support version 1.0 or 1.3 of this protocol. We note that data plane forwarding performance is not in the scope of this paper. Our contributions are as follows: (*i*) we advance the state-of-the-art in measuring OpenFlow switch control plane performance and its interaction with the data plane (for example, we dissect rule installation latency in a number of scenarios that bring the switch to the limit), (*ii*) we devise a more systematic way of switch testing, *i.e.*, along many different dimensions, than the existing work, and (*iii*), to be best of our knowledge, this is the first study to report several new types of anomalous behavior in OpenFlow switches. To further foster OpenFlow switch measurements and improvements to our work, we have made our tool publicly available at `https://bitbucket.org/bitnsg/switch-benchmark/wiki/Home`. Our tool was already adopted at a large European IXP while in the process of testing and deploying SDN.

Our key findings are as follows: (*i*) control plane performance is widely variable, and it depends on flow table occupancy, priorities, size of batches and even rule update patterns. In particular, priorities can cripple performance; (*ii*) switches might periodically or randomly stop processing control plane commands for up to 500 ms; (*iii*) data plane state might not reflect the control plane—it might fall behind by several minutes and it might also manifest rule installations in a different order than issued; (*iv*) seemingly atomic data plane updates might not be atomic at all. We summarize all findings and reference the section describing each of them in Table 1. By including new experiments and three new switches, this manuscript extends our previous findings by: (*i*) showing a new inconsistency pattern where data plane and control plane state divergence is unbounded; (*ii*) showing the variable characteristics of this divergence; (*iii*) showing that firmware is responsible for some issues by reporting our findings to a vendor and testing a fixed version; (*iv*) measuring that correct barrier handling is time consuming and affects switch update performance; (*v*)

confirming that different rule priorities slow down even a high-end switch.

The impact of our findings is multifold and profound. The non-atomicity of seemingly atomic data plane updates means that *there are periods when the network configuration is incorrect despite looking correct from the control plane perspective*. Existing tools that check if the controlplane is correctly configured [21, 22, 23] are unable to detect these problems. Moreover, the data plane can fall behind and unfortunately *barriers cannot be trusted*. This means that approaches for performing consistent updates need to devise a different way of defining when a rule is installed; otherwise they are not providing any firm guarantee. Finally, because the performance of a single switch depends on previously applied updates, developers need to account for this variable performance when designing their controllers.

The benefits of our work are numerous. First, we hope that SDN controller and framework developers will find our findings useful when trying to ensure consistent performance and reliability despite the variety of switches they might encounter. Thus, we report most of our findings with these developers in mind. For example, the existence of performance anomalies underlies the difficulty of computing an offline schedule for installing a large number of rules. Second, our study should serve as a starting point to measurement researchers to develop more systematic switch performance testing frameworks (*e.g.*, that have the ability to examine a large number of possible scenarios and pinpoint anomalies). Reporting findings presented in this paper to switch vendors has already helped them to detect bugs and improve the switch firmware. Third, efforts that are modeling switch behavior [15], should consult our study to become aware of the difficulty of precisely modeling switch performance.

Finally, we do not want to blame anyone and we know that OpenFlow support is sometimes provided as an experimental feature in the switches. The limitations we highlight should be treated as a hint where interesting research problems lay. If these problems still exist after several years of development, they may be caused by limitations that are hard or impossible to overcome, and could be present in the future switch generations as well. An example of such a well known limitation, unrelated to performance, is the flow table size. Researchers and switch developers understand that big TCAMs are expensive and thus try to save space in various ways [24, 25, 26, 27].

The remainder of the paper is organized as follows. Section 2 presents background and related work. We describe our measurement methodology in Section 3. We discuss in detail our findings about the data and control planes in Section 4 and show additional update rate measurements in Section 5.

2. Background and related work

SDN is relatively young, and therefore we first introduce the domain and explain the terminology used in this paper. We present SDN as realized by the OpenFlow protocol — currently the most popular implementation of SDN. The main idea behind SDN is to separate the switch data plane, that forwards packets, from the control plane, that is responsible for configuring the data plane. The control plane is further physically distributed between a switch and a controller running on a general-purpose computer (or cluster for reliability). The controller communicates with the switch to instruct it how to configure the data plane by sending flow modification commands that place rules in the switch's flow table. The switch-local control plane is realized by an OpenFlow agent — firmware responsible for the communication with the controller and for applying the updates to the data plane.

The controller generally needs to keep track of what rules the switch has installed in the data plane. Any divergence between the view seen by the controller and the reality may lead to incorrect decisions and, ultimately, wrong network configuration. However, the protocol does not specify any positive acknowledgment that an update was performed [28]. The only way to infer this information is to rely on the barrier command. As specified in the OpenFlow protocol [29], after receiving a barrier request, the switch has to finish processing all previously-received messages before executing any messages after the barrier request. When the processing is complete, the switch must send a barrier reply message.

Both data and control plane performance is essential for successful OpenFlow deployments, therefore it was a subject of measurements in the past. During their work on the FlowVisor network slicing mechanism, Sherwood *et al.* [17] report CPU-limited switch performance of about a few hundreds of OpenFlow port

status requests per second. Similarly, as part of their work on the DevoFlow modifications of the OpenFlow model, Curtis *et al.* [16] identify and explain the reasons for relatively slow rule installation rate on an HP OpenFlow switch. OFLOPS [18] is perhaps the first framework for structured OpenFlow switch evaluation. It combines a generic and open software framework with high-precision hardware instrumentation. OFLOPS performs fine-grained measurements of packet modification times, flow table update rate, and flow monitoring capabilities. This work was the first to make a number of important observations, for example that some OpenFlow agents did not support the barrier command. It was also the first work to report on the delay between the control plane's rule installation time and the data plane's ability to forward packets according to newly installed rules. OFLOPS-Turbo [30] is a continuation of this work that integrates with the Open Source Network Tester [31], which is built upon the NetFPGA platform to improve measurement precision even more. Huang *et al.* [15] perform switch measurements to construct high-fidelity switch models that may be used during emulation with the software-based Open vSwitch tool. Their work quantifies the variations in control path delays and the impact of flow table design (hardware, software, combinations thereof) at a coarse-grained level (average behavior). They also report surprisingly slow flow setup rates. Relative to these works, we dissect switch performance over longer time periods, and more systematically in terms of rule combinations, set of parameters, batch sizes, in-flight batch numbers, presence of barrier messages, and switch firmware versions. In addition, we identify thresholds that reveal previously unreported anomalous behaviors.

Several works have considered various issues that arise with diverse SDN switch hardware capabilities and ways to account for this diversity. A recent measurement study [32] focuses on data plane update rates. We observe both data and control planes and compare states in both. We also reveal performance variability present only in longer experiments. Jive [33] was perhaps the first proposal to build a proactive OpenFlow switch probing engine. Jive measures performance using predetermined patterns, *e.g.*, inserting a sequence of rules in order of increasing/decreasing priority, and reports large differences in installation times in an hardware switch. The observed switch behavior can be stored in a database, and later used to increase network performance. We show that the switch performance depends on so many factors that such a database would be difficult to create. Tango [19] proposed a proactive probing engine that infers key switch capabilities and behaviors according to well-structured patterns. It uses the probing results to perform automatic switch control optimization. Our study contributes a methodology that can be used to enrich the types of inferences used in this approach. NOSIX [34] notices the diversity of OpenFlow switches and creates a layer of abstraction between the controller and the switches. The idea is to be able to offer a portable API whose implementation makes use of commands optimized for a particular switch based on its capabilities and performance. However, the authors do not analyze dynamic switch properties as we do. We believe our work would be useful for NOSIX to improve the optimization process.

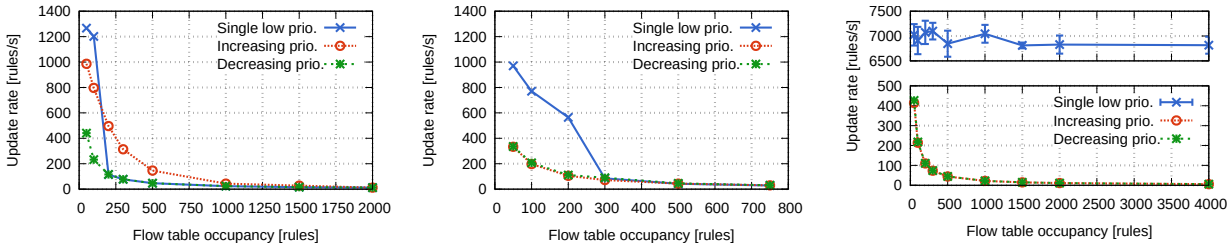
Finally, this paper adds many new results and insights to our previous work on the same topic [20] as we have elaborated earlier.

3. Measurement methodology

This section describes the methodology we follow to design the benchmarks that assess control and data plane update performance of switches under test.

3.1. Tools and experimental setup

In this study we focus on two metrics describing switch behavior: flow table rule update rate and correspondence between control plane and data plane views. The second metric is quantified by the time gap between when the switch confirms a rule modification and when the modified rule starts affecting packets. We designed a general methodology that allows for systematic exploration of switch behaviors under various conditions. At the beginning of each experiment, we prepopulate the switch flow table with R rules. Unless otherwise specified, the rules are non overlapping and have the default priority. Each rule matches a flow based on a pair of IP source-destination addresses, and forwards packets to switch port α. For clarity, we identify flows using contiguous integer numbers starting from $-R + 1$. According to this



(a) Pica8 P-3290 (b) Dell 8132F (c) NoviSwitch 1132

Figure 14: Switch rule update performance for different rule priority patterns.

update rates from the data plane experiments. As previously explained, there are three phases in this switch operation: slow rate when the switch is busy with control plane, fast rate when the switch does not deal with the control plane, and a very slow phase after the switch has installed about 4600 rules. Table 7 contains update rates in these three phases depending on the flow table occupancy (phase II is missing when the transition to phase III happens before all control plane messages are processed, phase III is missing for 7% occupancy, because the experiment is too short to reveal it). The results show that the switch performs similarly to other tested devices (Figure 12) until it installs 4600 rules during the experiment. After that point the performance drops significantly (phase III). It is also visible that the switch can modify rules two times quicker when it does not need to process control plane messages (phase II).

Summary: The performance of most tested switches drops with a number of installed rules, but the absolute values and the slope of this drop vary. Therefore, controller developers should not only take into account the total flow table size, but also what is the performance cost of filling the table with additional rules.

5.3. Priorities decrease the update rate

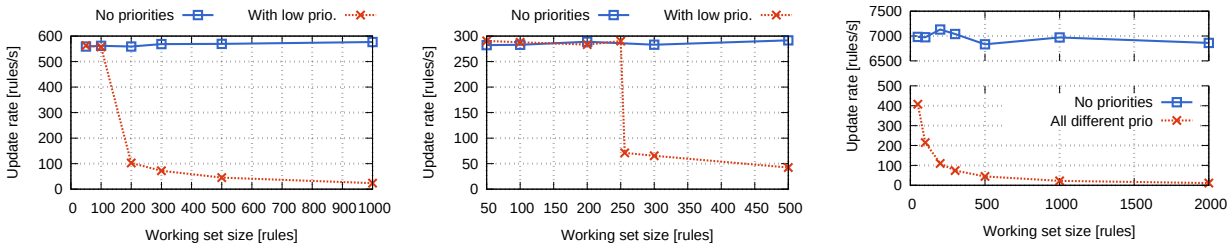
OpenFlow allows to assign a priority to each rule, but all our previous experiments considered only rules with equal, default priorities. A packet is always processed according to the highest priority rule that matches its header. Furthermore, in OpenFlow 1.0, the default behavior for a packet not matching any rule is to encapsulate it in a PacketIn message and send to the controller. To avoid overloading the controller, it is often desirable to install a lowest priority all-matching rule that drops packets. We conduct an experiment that mimics such a situation. The experiment setup is exactly the same as the one described in Section 5.2 with one additional lowest priority drop-all rule installed before all flow-specific rules.

Figure 13 shows that for a low flow table occupancy, all switches perform the same as without the low priority rule. However, Pica8 P-3290 and Dell 8132F suffer from a significant drop in performance at about 130 and 255 installed rules respectively. After this massive drop, the performance gradually decreases until it reaches 12 updates/s for 2000 rules in the flow table for Pica8 P-3290 and 30 updates/s for 750 rules in the flow table for Dell 8132F where both switches have their tables almost full. Interestingly, HP 5406zl's update rate does not decrease, possibly because it ignores the priorities. Switch Y and NoviSwitch 1132 update their flow tables at the same rate with and without the low priority rule. Again, for plot readability we do not show the rate for NoviSwitch 1132, which is an order of magnitude higher than other switches. We confirm that the results are not affected by the fully wildcarded match or the drop action in the low priority rule by replacing it with a specific IP src/dst match and a forwarding action.

Finally, we rerun the experiments from Section 5.1 with a low priority rule. The rates for Pica8 P-3290 and Dell 8132F are lower, but the characteristics and the conclusions hold.

More priorities: Next, we check the effect of using different priorities for each rule. We modify the default set-up such that each rule has a different priority assigned and install them in an increasing (rule i has a priority $D + i$, where D is the default priority value) or decreasing (rule i has a priority $D - i$) order.

Switches react differently. As it is visible in Figure 14, both Pica8 P-3290's and Dell 8132F's performance follows a similar curve as in the previous experiment. There is no breaking point though. In both cases the



(a) Pica8 P-3290 (b) Dell 8132F (c) NoviSwitch 1132

Figure 15: Size of the rule working set size a□ects the performance. For both Pica8 P-3290 and Dell 8132F when the low priority rule is installed, the performance depends mostly on the count of the rules being constantly changed and not on the total number of rules installed (1000 for Pica8 P-3290 and 500 for Dell 8132F in the plots). The same can be said about NoviSwitch 1132 with various rule priorities (2000 installed rules in the plot).

Priorities	1000 rules	2000 rules
$D - \frac{i}{10}$	216 rules/s	110 rules/s
$D - \frac{i}{20}$	374 rules/s	215 rules/s
$D - (i\%10)$	5222 rules/s	5588 rules/s
$D - (i\%20)$	6468 rules/s	6142 rules/s

Table 8: Flow table update rate in NoviSwitch 1132 depending on priority patterns and flow table occupancy. The rate depends on the number of priorities in use and number of newly added priorities.

performance is higher with only a single di□erent priority rule until the breaking point, after which they become equal. Further, Pica8 P-3290 updates rules quicker in the increasing priority scenario.[7]

Figure 14 shows that also NoviSwitch 1132 becomes significantly slower when there are additional priorities used as the update rate depends on the number of rules in the flow table. Even with just 50 installed rules, the rate drops from original 7000 updates/s to about 420. When the table occupancy increases the rate is as low as 5 updates/s. Update patterns does not matter – in the decreasing priority scenario the rate is minimally higher (up to 3%). In both cases, the update rate is inversely proportional to the occupancy. A deeper analysis shows, that the rate depends more on the number of priorities used than a total number of rules (Table 8). For example, the rate with 1000 rules in the table when rule i has a priority $D - \frac{i}{10}$ is almost equal to the rate with 100 initial rules in Figure 14. Further, it also seems that adding a rule with a new priority to the table takes a lot of time. When we run the experiment with rules using the same priorities as rules installed in the table before the experiment started, the rate is much higher. The vendor confirms that handling many priorities requires the switch to move some rules in TCAM, which makes updates slower. They use optimizations to reduce the impact of move operations when the number of priorities is small.

HP 5406zl control plane measurement is not a□ected by the priorities, but as our data plane study shows there is a serious divergence between the control plane reports and the reality for this switch in this experiment (see Section 6). Finally, using di□erent priorities does not a□ect Switch Y performance.

Working set size: Finally, we check what happens if only a small subset of rules in the table (henceforth referred to as "working set") is frequently updated. We modify the default experiment setup such that batch i deletes the rule matching flow number $i - W$ and installs a rule matching flow i. We vary the value of W. In other words, assuming there are R rules initially in the flow table, the first $R - W$ rules never change and we update only the last W rules.

The results show that HP 5406zl performance is una□ected and remains the same as presented in Figures 12 and 13 both below and above the threshold of 760 rules in the flow table. Further, for both Pica8 P-3290

[7] This is consistent with the observation made in [33], but the di□erence is smaller as for each addition we also delete the lowest priority rule.

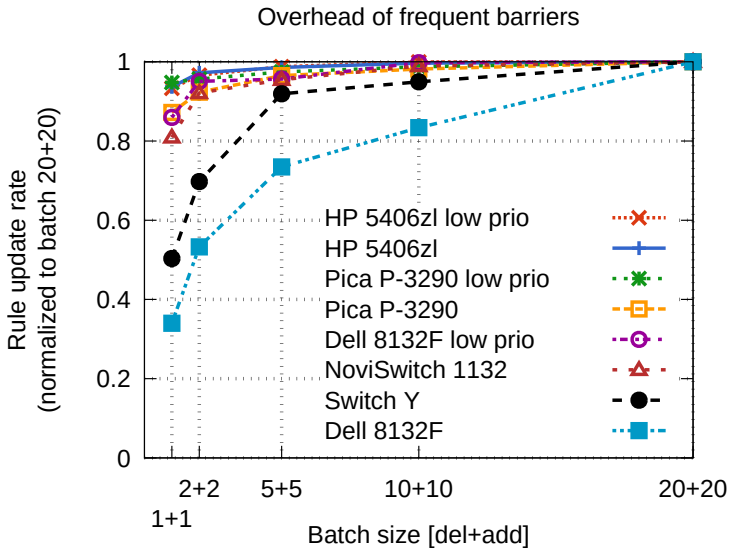


Figure 16: Cost of frequent barriers is modest except for the case of Dell 8132F with no priorities (*i.e.*, with high baseline speed) and Switch Y where the cost is significant.

and Dell 8132F a small working set for updates makes no difference if there is no low priority rule. For a given R (1000 for Pica8 P-3290 and 500 for Dell 8132F in Fig. 15), the performance is constant regardless of W. However, when the low priority rule is installed, the update rate characteristic changes as shown in Figure 15. For both switches, as long as the update working set is smaller than their breaking point revealed in Section 5.2, the performance stays as if there was no drop rule. After the breaking point, it degrades and is only marginally worse compared to the results in Section 5.2 for table occupancy W.

A working set size affects NoviSwitch 1132 as well. In this case, we analyze its performance when using multiple priorities (Figure 15) with $R = 2000$. The rate depends on the working set size and is almost the same as the rate with the same total number of rules in the flow table.

Summary: The switch performance is difficult to predict—a single rule can degrade the update rate of a switch by an order of magnitude. Controller developers should be aware of such behavior and avoid potential sources of inefficiencies.

5.4. Barrier synchronization penalty varies

A barrier request-reply pair of messages is very useful, as according to the specification, it is the only way for the controller to (*i*) force an order of operations on the switch, and (*ii*) make sure that the switch control plane processed all previous commands. The latter becomes important if the controller needs to know about any errors before continuing on with the switch reconfiguration. Because barriers might be needed frequently, in this experiment we measure the overhead given a frequency with which we use barriers.

We repeat our general experiment setup with $R = 300$ preinstalled rules, this time varying the number of rule deletions and insertions in a single batch. To keep flow table size from diverging during the experiment, we use an equal number of deletions and insertions.

As visible in Figure 16, for both Pica8 P-3290 and HP 5406zl the rate slowly increases with growing batch size, but the difference is marginal: up to 14% for Pica8 P-3290 and up to 8% for HP 5406zl for a batch size growing 20 times. On the other hand, Dell 8132F speeds up 3 times in the same range if no priorities are involved. The same observation can be made for Switch Y.

While further investigating these results, we verified that the barrier overhead for each particular switch recalculated in terms of milliseconds is constant across a wide range of parameters – a barrier takes roughly 0.1-0.3ms for Pica8 P-3290, 3.1-3.4ms for Dell 8132F, 1ms for Switch Y, 0.6-0.7ms for HP 5406zl and 0.04ms

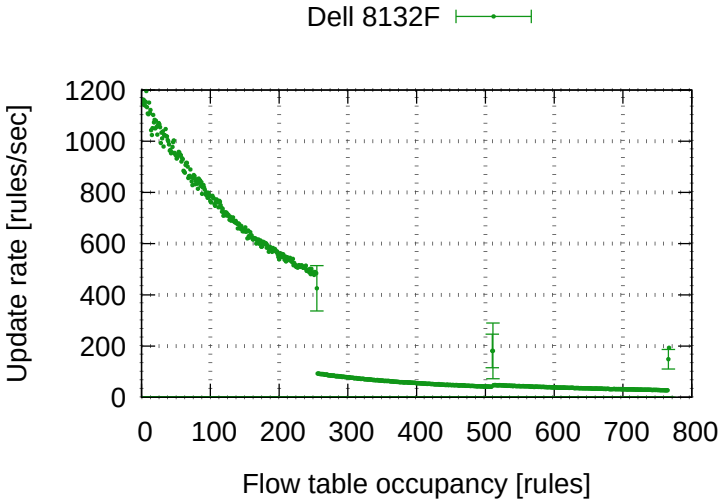


Figure 17: An update rate in Dell 8132F suddenly increases for 4 specific flow table occupancy values.

for NoviSwitch 1132. This explains the high overhead of Switch Y and Dell 8132F for fast rule installations in Figure 16 – barriers just take time comparable to rule installations. Taking into account that Switch Y and Dell 8132F are the only tested ASIC-based switches that provide correct barriers, our conclusion is that a working barrier implementation is costly.

Summary: Overall, we see that barrier cost varies across devices. The controller, therefore, should be aware of the potential impact and balance between the switch performance and potential notification staleness. Moreover, there is a tradeo between correct barrier implementation and performance.

6. Open questions and future work

In the process of running the experiments and gaining an understanding of the root causes of various unexpected behaviors, we made additional observations. We briefly report them in this section as this information may be useful or inspiring to investigate certain open issues further.

Rule insertion may act as a modification. In our experiments, we observed that two out of six switches are unable to perform an atomic rule modification. However, when receiving a rule insertion command for a rule that has the same match and priority as an already installed one, but a di erent set of actions, all the tested switches modify the existing rule. Moreover, this operation does not lead to any packet drops on HP 5406zl, which is better than the behavior obtained by using a rule modification command (Section 4.4). The behavior of Dell 8132F remains unchanged. We note that the OpenFlow specifications describe the behavior when a rule insertion command references an existing rule with identical match fields and priority. However, the behavior is to clear the existing rule and insert the new one. The fact that for HP 5406zl this operation works better than a modify command is surprising.

Data plane tra c can increase the update rate of Pica8 P-3290. We noticed that in some cases, sending data plane tra c that matches currently installed rules at Pica8 P-3290 can speed up the general update rate and even future updates. Our experiments show that barrier inter-arrival times (time between barrier replies for two consecutive barriers) shorten after the switch starts processing packets belonging to already installed flows. We confirmed that the behavior is consistent across varying flow ranges and long data series, however, we are unable to provide an explanation of this phenomenon at this time nor confirm it with full certainty. We find this completely counter-intuitive and leave it as an open question for future work.

Dell 8132F performs well with a full flow table. In Section 5.3, we reported that the performance of Dell 8132F with a low priority rule installed decreases with the growing table occupancy and drops down

to about 30 updates per second when the flow table contains 751 rules. We showed the update rate measured for all possible flow table occupancies in an experiment with 2000 update batches in Figure 17. We observed that this trend continues until the table is full or there is one slot left. Surprisingly, the switch performs updates that remove a rule and install a new one with a full table at a rate comparable to that observed without the low priority rule. There is also an unexpected sudden performance improvement at 510 and 511 rules. Measurements in both these points have a very high standard deviation, but the results for a full table are stable. Dell 8132F is a conventional switch adapted to support OpenFlow. According to its documentation, the switch contains two separate tables with space for 256 and 512 rules respectively. These numbers align well with performance changes we observe.

7. Conclusions

In this paper we try to shed light on the state of OpenFlow switches – an essential component of relatively new, but quickly developing Software Defined Networks. While we do our best to make the study as broad and as thorough as possible, we observe that the switch performance is so unpredictable and depends on so many parameters that we expect to reveal just the tip of the iceberg. However, even the observations made here should be an inspiration to revisit many assumptions about OpenFlow and SDN in general. The main takeaway is that despite a common interface, the switches are more diverse than one would expect, and this diversity has to be taken into account when building controllers.

Because of the limited resources, we managed to obtain sufficiently long access only to six switches over the years. In the future, we plan to keep extending this study with additional devices, as well as switches that are using alternative technologies (NetFPGA, network processors, etc.), to obtain the full picture. Measuring the precise data plane forwarding performance is another unexplored direction.

Acknowledgments We would like to thank Dan Levin and Miguel Peón for helping us get remote access to some of the tested switches. We also thank the representatives of the Pica8 P-3290, NoviSwitch 1132 and Switch X vendors for their quick and extensive responses that helped us understand some observations we made.

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/ 2007-2013) / ERC grant agreement 259110. This research is (in part) supported by European Union's Horizon 2020 research and innovation programme under the ENDEAVOUR project (grant agreement 644960). This work is in part financially supported by the Swedish Foundation for Strategic Research.

References

[1] Ethernet Switch Market: Who's Winning?, http://www.networkcomputing.com/networking/ ethernet-switch-market-whos-winning/d/d-id/1234913 (2014).

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, B4: Experience with a Globally-Deployed Software Defined WAN, in: SIGCOMM, 2013.

[3] TechTarget, Carriers bet big on open SDN, http://searchsdn.techtarget.com/news/4500248423/Carriers-bet-big-on-open-SDN, last visited on Oct 5, 2015.

[4] A. Greenberg, Microsoft Showcases Software Defined Networking Innovation at SIGCOMM, https://azure.microsoft.com/en-us/blog/microsoft-showcases-software-defined-networking-innovation-at-sigcomm-v2/ (August 2015).

[5] OpenDaylight, http://www.opendaylight.org/ (2014).

[6] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. i. Snow, G. Parulkar, ONOS: Towards an Open, Distributed SDN OS, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, 2014.

[7] S. H. Yeganeh, Y. Ganjali, Beehive: Simple Distributed Programming in Software-Defined Networks, in: Proceedings of the Symposium on SDN Research, SOSR '16, 2016.

[8] N. P. Katta, J. Rexford, D. Walker, Incremental Consistent Updates, in: HotSDN, 2013.

[9] R. Mahajan, R. Wattenhofer, On Consistent Updates in Software Defined Networks, in: HotNets, 2013.

[10] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, D. Walker, Abstractions for Network Update, in: SIGCOMM, 2012.

[11] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, D. A. Maltz, zUpdate : Updating Data Center Networks with Zero Loss, in: SIGCOMM, 2013.

[12] P. Perešíni, M. Kuźniar, M. Canini, D. Kostić, ESPRES: Transparent SDN Update Scheduling, in: HotSDN, 2014.

[13] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, R. Wattenhofer, Dynamic scheduling of network updates, in: SIGCOMM, 2014.
[14] T. D. Nguyen, M. Chiesa, M. Canini, Decentralized Consistent Updates in SDN, in: Proceedings of the Symposium on SDN Research, SOSR '17, 2017.
[15] D. Y. Huang, K. Yocum, A. C. Snoeren, High-fidelity switch models for software-defined network emulation, in: HotSDN, 2013.
[16] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, DevoFlow: Scaling Flow Management for High-Performance Networks, in: SIGCOMM, 2011.
[17] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, Can the Production Network Be the Testbed?, in: OSDI, 2010.
[18] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A. W. Moore, Oflops: An open framework for openflow switch evaluation, in: PAM, 2012.
[19] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, Y. R. Yang, M. Yu, Tango: Simplifying SDN Control with Automatic Switch Property Inference, Abstraction, and Optimization, in: CoNEXT, 2014.
[20] M. Kuźniar, P. Perešíni, D. Kostić, What You Need to Know About SDN Flow Tables, in: PAM, 2015.
[21] A. Khurshid, X. Zou, W. Zhou, M. Caesar, P. B. Godfrey, VeriFlow: Verifying Network-Wide Invariants in Real Time, in: NSDI, 2013.
[22] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, S. Whyte, Real Time Network Policy Checking using Header Space Analysis, in: NSDI, 2013.
[23] P. Kazemian, G. Varghese, N. McKeown, Header Space Analysis: Static Checking for Networks, in: NSDI, 2012.
[24] N. Kang, Z. Liu, J. Rexford, D. Walker, Optimizing the "One Big Switch" Abstraction in Software-Defined Networks, in: CoNEXT, 2013.
[25] N. Katta, O. Alipourfard, J. Rexford, D. Walker, CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks, in: SOSR, 2016.
[26] X. Wen, B. Yang, Y. Chen, L. E. Li, K. Bu, P. Zheng, Y. Yang, C. Hu, RuleTris: Minimizing Rule Update Latency for TCAM-Based SDN Switches, in: ICDCS, 2016, pp. 179–188.
[27] H. Chen, T. Benson, The Case for Making Tight Control Plane Latency Guarantees in SDN Switches, in: SOSR, 2017.
[28] P. Perešíni, M. Kuźniar, D. Kostić, OpenFlow Needs You! A Call for a Discussion about a Cleaner OpenFlow API, in: EWSDN, IEEE, 2013.
[29] OpenFlow Switch Specification, http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf.
[30] C. Rotsos, G. Antichi, M. Bruyere, P. Owezarski, A. W. Moore, OFLOPS-Turbo: Testing the next-generation OpenFlow switch, in: IEEE ICC, 2015.
[31] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. McKeown, N. Feamster, B. Felderman, M. Blott, A. W. Moore, P. Owezarski, OSNT: Open Source Network Tester, IEEE Network 28 (5) (2014) 6–12.
[32] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, M. Thottan, Measuring control plane latency in SDN-enabled switches, in: SOSR, 2015.
[33] A. Lazaris, D. Tahara, X. Huang, L. E. Li, A. Voellmy, Y. R. Yang, M. Yu, Jive: Performance Driven Abstraction and Optimization for SDN, in: ONS, 2014.
[34] M. Yu, A. Wundsam, M. Raju, NOSIX: A Lightweight Portability Layer for the SDN OS, ACM SIGCOMM Computer Communication Review 44 (2).
[35] NOX Controller, http://github.com/noxrepo/nox.
[36] ROFL Library, http://roflibs.org.

Maciej Kuzniar obtained a Ph.D. degree from EPFL in 2016 and a Master of Science degree in Computer Science from AGH University of Science and Technology in Krakow in 2011. His research focuses on Software Defined Networks, especially aspects a□ecting their reliability. Additionally, he is interested in the wider area of distributed systems and computer networks. He currently works at Google.

Peter Perešíni's research interests include Software-Defined Networking (with the focus on correctness, testing, reliability, and performance), Distributed Systems, and Computer Networking in general. He obtained his Ph.D. from EPFL, Switzerland in 2016 and a Master of Science degree with distinction from Comenius University in Bratislava in 2011. Peter completed multiple internships at Google, working on di□erent projects.

Dejan Kostic is a Professor of Internetworking at the KTH Royal Institute of Technology, where he is the head of the Network Systems Laboratory. His research interests include Distributed Systems, Computer Networks, Operating Systems, and Mobile Computing. Dejan Kostic obtained his Ph.D. in Computer Science at the Duke University. He spent the last two years of his studies and a brief stay as a postdoctoral scholar at the University of California, San Diego.

Marco Canini is an assistant professor of computer science at King Abdullah University of Science and Technology (KAUST). His research interests include software-defined networking and large-scale and distributed cloud computing. He received a Ph.D. in computer science and engineering from the University of Genoa. He is a member of IEEE, ACM and USENIX.