Computer Networks 136 (2018) 22–36

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches

Maciej Kuźniar[a,1], Peter Perešíni[a,2], Dejan Kostić[c,*], Marco Canini[b]

[a] EPFL, Switzerland
[b] KAUST, Saudi Arabia
[c] KTH Royal Institute of Technology, Sweden

ARTICLE INFO

Article history:
Received 30 June 2017
Revised 25 December 2017
Accepted 14 February 2018
Available online 15 February 2018

Keywords:
Software-defined networking
Switch
Flow table updates
Measurements

ABSTRACT

Software-Defined Networking (SDN) and OpenFlow are actively being standardized and deployed. These deployments rely on switches that come from various vendors and differ in terms of performance and available features. Understanding these differences 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 efficient. Further, we also highlight differences between the OpenFlow specification and its implementations, that if ignored, pose a serious threat to network security and correctness.

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 [1–4]. This means that the number of SDN developers creating exciting new frameworks [5–7] 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 [8–10] 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 [9,12–14]. 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.

Initially, sporadic OpenFlow switch performance measurements were reported [15–17]. 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

* Corresponding author.
E-mail addresses: dmk@kth.se, dkostic@acm.org (D. Kostić).
[1] Google.
[2] Unaffiliated.

https://doi.org/10.1016/j.comnet.2018.02.014
1389-1286/

Table 1
Summary of key findings presented in this paper.

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.

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 control plane is correctly configured [21–23] 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 [24–27].

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. Lazaris et al. [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. Lazaris et al. [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 notation, the prepopulated rules match flows in the range $-R+1$ to 0, inclusive. The initial setup rules have negative numbers so that the main experiment rules start from 1.

After initializing the switch's hardware flow table, we perform flow table updates and measure their behaviors. In particular, we send B batches of rule updates, each batch consisting of: B_D rule deletions, B_M rule modifications and B_A rule insertions. Each batch is followed by a barrier request. Depending on the experiment, we adjust the number of in-flight batches. The controller sends a new batch only if the switch did not send a reply for at most a given number of previously sent barriers. In the default setup, we set $B_D = B_A = 1$ and $B_M = 0$. If B_D is greater than 0, batch i deletes rules matching flows with numbers between $-R+1+(i-1)*B_D$ and $-R+i*B_D$. If B_A is greater than 0, batch i installs rules that match flows with numbers in range between $(i-1)*B_A+1$ and i^*B_A and forwards packets to port α. As a result, each batch removes the oldest rules. Note that the total number of rules in the table remains stable during most experiments (in contrast to previous work such as [33] and [18] that measure only the time needed to fill an empty table).

To measure data plane state, in some experiments, we inject and capture data plane traffic. We send packets that belong to flows F_{start} to F_{end} (inclusive) at a rate of about 100,000 packets per second (which translates to about 1000 packets per flow per second).

In our study, we have explored a wide range of possible parameters for our methodology. For brevity, in the next sections, we highlight results where we instantiate the methodology with specific parameters that led to interesting observations. In the experiment descriptions we call the setup described above with $B_D = B_A = 1$, $B_M = 0$ and all rules with equal priority as a general experimental setup. Finally, unless an experiment shows variance greater

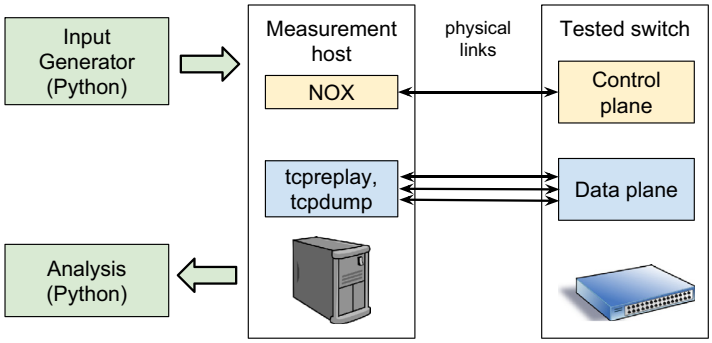


Fig. 1. Overview of our measurement tools and testbed setup.

than 5% across runs, we repeat it three times and report the average. Because the results have a small deviation across runs, unless otherwise specified, we do not show confidence intervals.

Measurement tool: Based on our initial investigation, as well as previously reported results [15], we identify three main requirements for a measurement tool: (*i*) Flexibility, (*ii*) Portability, and (*iii*) Sufficient precision. Our previous experience suggests that switches behave unexpectedly, and thus we need to tailor the experiments to locate and dissect problems. Moreover, as the tested switches can modify at most a few thousands of rules per second, we assume that a millisecond measurement precision is sufficient.

To achieve the aforementioned goals, we built a tool that consists of three major components that correspond to the three benchmarking phases: input generation, measurement and data analysis (Fig. 1).

First, an input generator creates control plane rule modification lists and data plane packet traces used for the measurements. Unless otherwise specified, the forwarding rules used for the experiments match traffic based on IP source/destination pairs and forward packets to a single switch port. Moreover, we notice that some switches can optimize rule updates affecting the same rule; we therefore make sure that modifications affect different rules. To ensure this, by default, we use consecutive IPs for matches. Furthermore, we cross-check our results using random matches and update patterns.

We refer to the control plane measurement engine as the controller as it emulates the behavior of an OpenFlow controller. We implement it using NOX [35] and ROFL [36] libraries that can issue rule updates at a much higher rate than what the hardware switches can handle.[3] The engine records time of various interactions with the switch (*e.g.*, flow modification sent, barrier reply received) and saves all its outputs into files. We additionally record all control plane traffic using tcpdump. We rely on existing tcpreplay and tcpdump tools to both send packets based on a pcap file and record them. This way we ensure that packets flow only in one direction and have a single interaction with a switch. To remove time synchronization issues, we follow a simple testbed setup with the switch connected to a single host on multiple interfaces — the host handles the control plane as well as generates and receives traffic for the data plane. Note that we do not need to fully saturate the switch data plane, and thus a conventional 48-core host is capable of handling all of these tasks at the same time.

Finally, a modular analysis engine reads the output files and computes the metrics of interest. Modularity means that we can add a new module to analyze a different aspect of the measured data. We implement the analysis engine as a collection of modules code in Python.

Switches under test: We benchmark three ASIC-based switches capable of OpenFlow 1.0 and two ASIC-based switches capable of OpenFlow 1.3 support: HP ProCurve 5406zl with K.15.10.0009 firmware, Pica8 P-3290 with PicOS 2.0.4, Dell PowerConnect 8132F with beta[4] OpenFlow support, Switch X and Switch Y. They use ProVision, Broadcom Firebolt, Broadcom Trident+, Switch X and Switch Y ASICs, respectively. We additionally compare how Switch X behaves with two firmware versions: V1 and V2. We anonymize two of the switches since we did not get a permission to use their names from their respective vendors. We note that Switch Y is a high-end switch. These switches have two types of forwarding tables: hardware and software. The switches have various hardware flow table sizes: about 1500, 2000, 750, 4500, and 2000 rules, respectively. While hardware table sizes and levels of OpenFlow support vary, we make sure that all test rules ultimately end up in hardware tables. Moreover, some switches implement a combined mode where packet forwarding is done by both hardware and software, but this imposes high load on the switch's CPU and provides lower forwarding performance. Thus, we avoid studying this operating mode. Further, as mentioned before, analyzing the data plane forwarding performance is also out of scope of this paper. We also benchmark NoviSwitch 1132— a high-end network-processor based, OpenFlow 1.3 switch running firmware version 300.0.1.[5] Each of its 64 flow tables fits over 4000 rules. We caution that the results for this switch may not directly compare to those of the other measured devices due to the different switch architecture. In particular, our methodology correctly characterizes the update rates of flow tables but does not establish a relation between flow table occupancy and maximum forwarding speed, for which ASICs and network processor might exhibit different behaviors.

Finally, since the switches we tested are located in different institutions, there are small differences between the testing machines and the network performance. However, the setups are comparable. A testing computer is always a server-class machine and the network RTT varies between 0.1 and 0.5 ms.

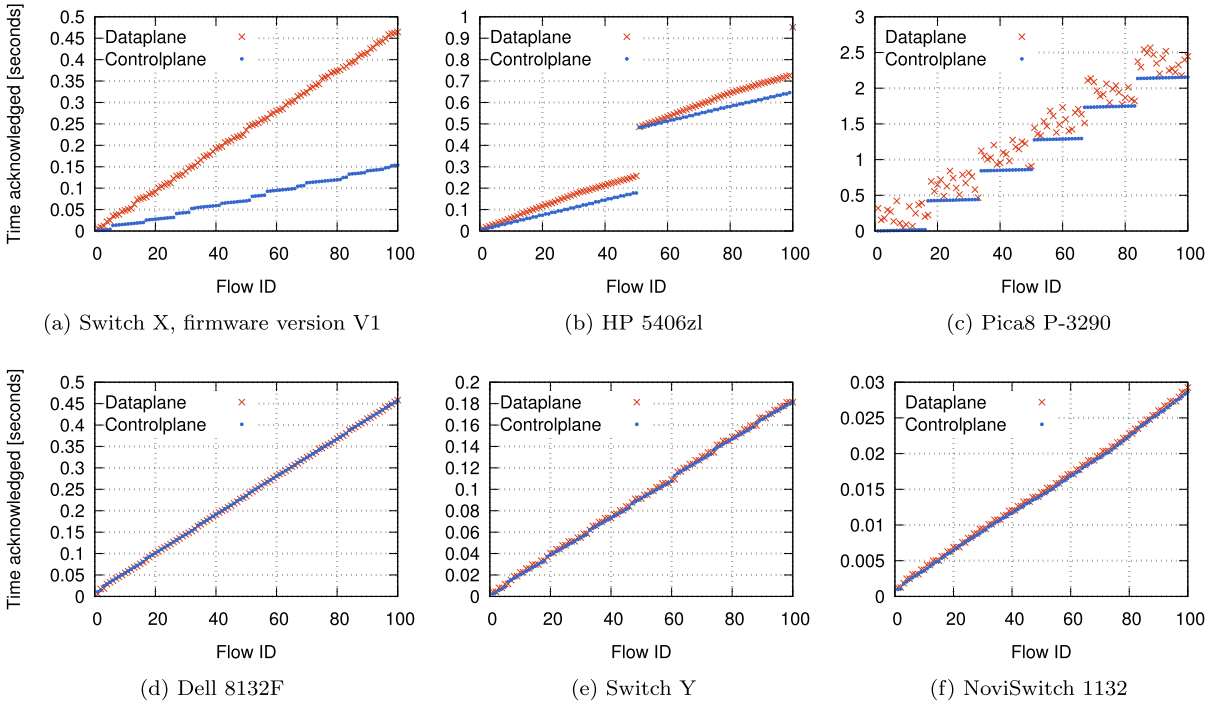
4. Flow table consistency

While the only view the controller has of the switch is through the control plane, the real traffic forwarding happens in the data plane. In this section we present the results of experiments where we monitor rule updates in the control plane and at the same time send traffic to exercise the updated rules. The unexpected behavior we report in this section may have negative implications for network security and controller correctness.

[3] Our benchmark with software OpenVSwitch handles ~42, 000 updates/s.

[4] There are plans to optimize and productize this software.

[5] We repeated our tests with firmware 300.0.5 but observed similar results.



(a) Switch X, firmware version V1 (b) HP 5406zl (c) Pica8 P-3290

(d) Dell 8132F (e) Switch Y (f) NoviSwitch 1132

Fig. 2. Control plane confirmation times and data plane probe results for the same flows. Switch data plane installation time may fall behind the control plane acknowledgments and may be even reordered.

4.1. Synchronicity of control and data planes

Many solutions essential for correct and reliable OpenFlow deployments (*e.g.*, [10,11]) rely on knowing when the switch applied a given command *in the data plane*. The natural method to get such information is the barrier message.[6] Therefore, it is crucial that this message works correctly. However, as authors of [18] already hinted, the state of the data plane may be different than the one advertised by the control plane. Thus we set out to measure how these two views correspond to each other at a fine granularity.

We use the default setup extended with one match-all low priority rule that drops all packets[7] and we inject data plane flows number F_{start} to F_{end}. For each update batch i we measure the time when the controller receives a barrier reply for this batch and when the first packet of flow i reaches the destination.

Fig. 2 shows the results for $R = 300$, $B = 300$, $F_{start} = 1$ and $F_{end} = 100$. There are three types of behavior that we observe: desynchronizing data and control plane states, reordering rules despite barriers and correct implementation of the specification.

Switch X: The data plane configuration of *Switch X is slowly falling behind* the control plane acknowledgments – packets start reaching the destination long after the switch confirms the rule installation with a barrier reply. The divergence increases linearly and, in this experiment reaches 300 ms after only 100 rules. The second observation is that *Switch X installs rules in the order of their control plane arrival*. After reporting the problem of desynchronized data and control plane views to the switch vendor, we received a new firmware version that fixed observed issues to some extent. We report the improvements in Section 4.3.

HP 5406zl: Similarly to Switch X, the data plane configuration of *HP 5406zl is slowly falling behind* the control plane acknowledgments. However, unlike for Switch X, after about 50 batches, which corresponds to 100 rule updates (we observed that adding or deleting a rule counts as one update, and modifying an existing rule as two), the *switch stops responding with barrier replies for 300 ms*, which allows the flow tables to catch up. After this time the process of diverging starts again. In this experiment the divergence reaches up to 82 ms, but *can be as high as 250 ms* depending on the number of rules in the flow table. Moreover, the frequency and the duration of this period does not depend on the rate at which the controller sends updates, as long as there is at least one update every 300 ms. The final observation is that *HP 5406zl installs rules in the order of their control plane arrival*.

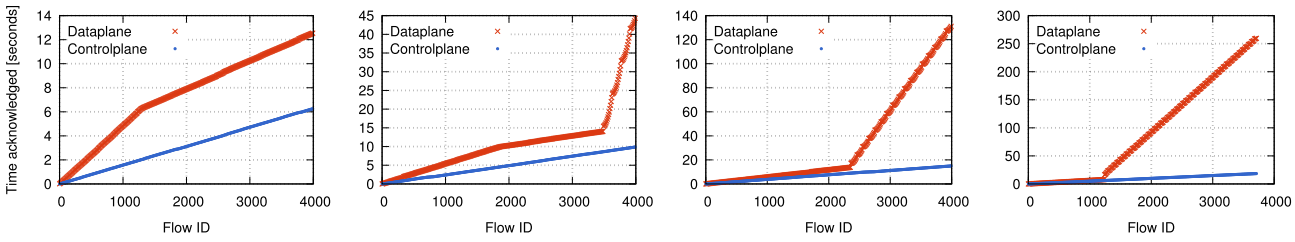
Pica8 P-3290: Similarly to HP 5406zl, Pica8 P-3290 stops responding to barriers in regular intervals. However, unlike HP 5406zl and Switch X, Pica8 P-3290 is either processing control plane (handling update commands and responding to barriers), or installing rules in TCAM and never does both at the same time. Moreover, *despite the barriers, the rules are not installed in hardware in the order of arrival*. The delay between data and control plane reaches *up to 400 ms* in this experiment. When all remaining rules get pushed into hardware, the switch starts accepting new commands in the control plane again. We confirmed with a vendor that because the synchronization between the software and hardware table is expensive, it is performed in batches and the order of updates in a batch is not guaranteed. When the switch pushes updates to hardware, its CPU is busy and it stops dealing with the control plane.[8]

Dell 8132F, Switch Y and NoviSwitch 1132: *All three switches make sure that no control plane confirmation is issued before a rule becomes active* in hardware. In this experiment we do not see any periods of idleness as the switch pushes rules to hardware all the

[6] As specified, 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 [29].

[7] We need to use such a rule to prevent flooding the control channel with the PacketIn messages caused by data plane probes or flooding the probes to all ports.

[8] The Vendor claims that this limitation occurs only in firmware prior to PicOS 2.2.



(a) 7% (300 rules) (b) 25% (1125 rules) (c) 50% (2250 rules) (d) 75% (3375 rules)

Fig. 3. Control plane confirmation times and data plane probe results for the same flows in Switch X (firmware version V1) depending on flow table occupancy. The rate suddenly slows down after about 4600 flow installations (including initial rules installed before the experiment starts).

Table 2
Data plane synchronicity key findings summary.

Switch	Data plane
Switch X, firmware V1	Falls behind indefinitely. Up to 4 minutes in our experiments.
Switch X, firmware V2	In sync with control plane
HP 5406zl	Often falls behind up to 250 ms. Indefinitely in corner cases (up to 22s in our tests).
Pica8 P-3290	Reorders + behind up to 400 ms
Dell 8132F	In sync with control plane
Switch Y	In sync with control plane
NoviSwitch 1132	In sync with control plane

time and waits for completion if necessary. Additionally, because NoviSwitch 1132 is very fast, we increased the frequency of sending data plane packets in order to guarantee required measurement precision. Table 2.

***Summary**: To reduce the cost of placing rules in a hardware flow table, vendors allow for different types (e.g., falling behind or reordering) and amounts (up to 400 ms in this short experiment) of temporary divergence between the hardware and software flow tables. Therefore, the barrier command does not guarantee flow installation. Ignoring this problem leads to an incorrect network state that may drop packets, or even worse, send them to an undesired destination!*

4.2. Variability in control and data plane behavior

The short experiment described in the previous section reveals three approaches to data and control plane synchronization. In this section we report more detailed unexpected switch behavior types observed when varying parameters in that experiment. The overall setup stays the same, but we modify the number of rules in the flow tables, length of the experiments and range of monitored rules.

Switch X: The short experiment revealed that Switch X never gives the data plane state a chance to synchronize with control plane acknowledgments. In this extended experiment we issue 4000 batches of rule deletion and rule installation and monitor every 10th rule. Fig. 3 shows the results for various flow table occupancy (7%, 25%, 50% and 75%). There are three main observations. First, the switch indeed does not manage to synchronize the control and data plane states. Second, the update rate increases when the switch is no longer busy with receiving and sending control plane messages. This is visible as a change of slope of the data plane line in Fig. 3(a) and (b). We confirmed this observation by sending additional echo or barrier messages. If the switch control plane stays busy, the data plane line grows at a constant rate. We believe a low power CPU used in this switch can easily become a bottleneck and cause the described behavior. Finally, after installing about 4600 rules since the last full table clear, the switch becomes significantly slower and the gap between what it reports in the control plane and its actual state quickly diverges. We kept monitoring the data plane for 4 minutes after the switch reported all rule modifications completed, and still not all rules were in place yet. We run additional tests and it seems that even performing updates at a lower rate (2 updates every 100 ms) or waiting for a long time (wait for 8s after every 200 updates) does not solve the problem. The risk is that *the switch performance may degrade in any deployment where the whole flow table is rarely cleared*. We reported aforementioned issues to the switch vendor and received a confirmation and an improved firmware version.

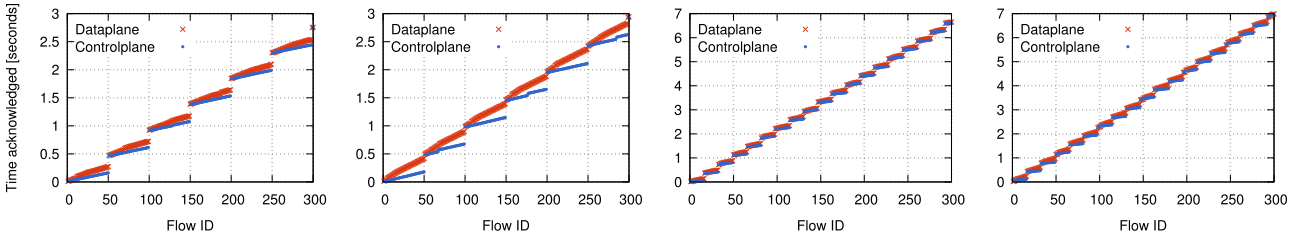
HP 5406zl: The pattern observed in the previous experiment does not change when parameters vary except for two details depending on the flow table occupancy. We show them in Fig. 4. First, the 300 ms inactivity time is constant across all the experiments, but happens three times more often (every 33 updates) if there are over 760 rules in the flow table (Fig. 4(c)). Second, when the number of rules in the flow table increases, the maximum delay between control and data plane update increases as well. It reaches 250 ms when there are 750 rules in the table (Fig. 4(b)). For over 760 rules, the switch synchronizes more frequently, so the maximum delay is smaller again (Fig. 4(c)) but goes back to 150 ms for 1300 rules (Fig. 4(d)). We conclude that *the real flow table update speed in HP 5406zl depends on the number of rules in the table*, and the switch accounts for a possible delay by letting the data plane to catch up in regular intervals.

However, we found cases when the switch does not wait long enough, which may lead to unlimited divergence between the data and control planes. First, in Fig. 5 we show that when different priorities are used (each rule has a different priority in this experiment), the switch becomes very slow in applying the changes in hardware without notifying the control plane. This behavior is especially counter-intuitive since the switch does not support priorities in hardware. Second, our experiments show that rule deletions are much faster than installations. Fig. 6 shows what happens when we install 500 rules starting from an empty flow table with only a single drop-all rule. Until there are 300 rules in the table, the 300 ms long periods every 100 updates are sufficient to synchronize the views. Later, the data plane modifications are unable to keep up with the control plane.

Pica8 P-3290: There are no additional observations related to Pica8 P-3290. The pattern from Fig. 2(c) occurs during the whole experiment.

Dell 8132F: As depicted in Fig. 7, the switch starts updating rules quickly, but suddenly slows down after 210 new rules installed and maintains this slower speed (verified up to 2000 batches). However, even after the slowdown, the control plane reliably reflects the state of the data plane configuration. Additionally, we observe periods when the switch does not install rules or respond to the controller, but these periods are rare, non reproducible and do not seem to be related to the experiments.

Switch Y: Although in the original experiment we observe no periods of idleness, when the flow table occupancy and the experiment running time increase, the switch stops processing requests



(a) 300 rules (b) 750 rules (c) 770 rules (d) 1300 rules

Fig. 4. Control plane confirmation times and data plane probe results for the same flows in HP 5406zl depending on flow table occupancy. The rate slows down and the pattern changes for over 760 rules in the flow table.

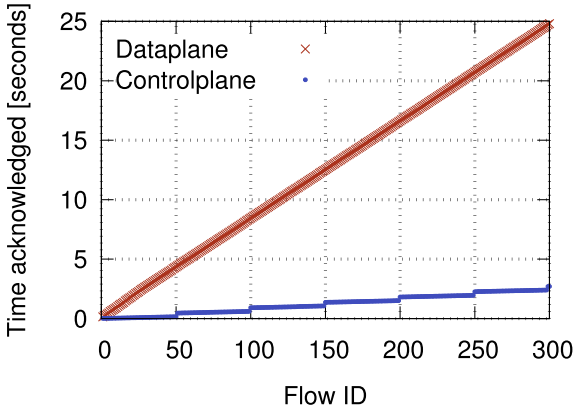


Fig. 5. Control plane confirmation times fall behind the data plane probe results in HP 5406zl when using rules with different priorities. The scale of divergence is unlimited.

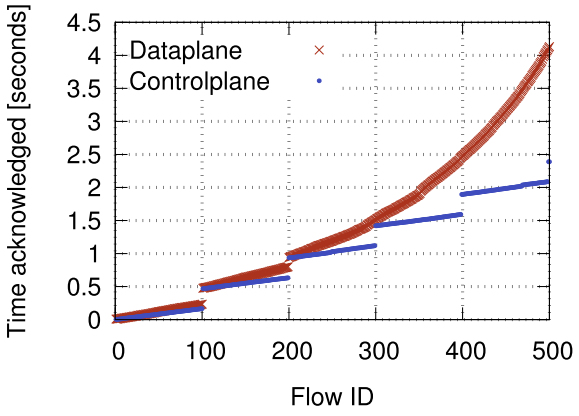


Fig. 6. Control plane confirmation times fall behind the data plane probe results in HP 5406zl when filling the flow table.

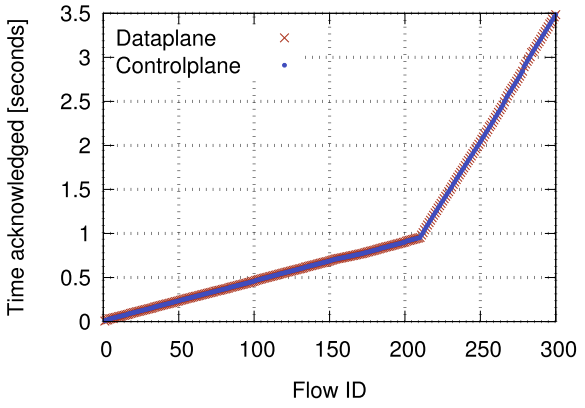


Fig. 7. Control plane confirmation times and data plane probe results in Dell 8132F are synchronized, but the update rate suddenly slows down after about 210 newly installed rules.

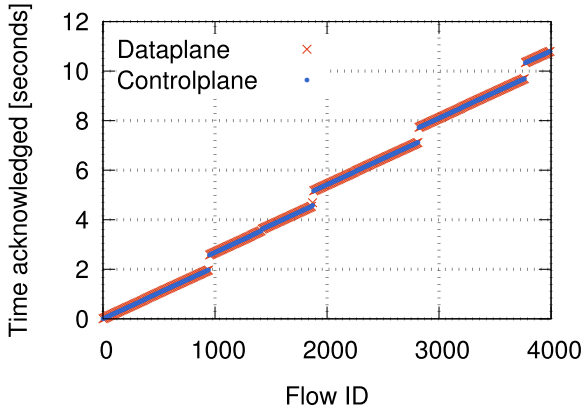


Fig. 8. Control plane confirmation times and data plane probe results in Switch Y with 95% table occupancy are synchronized, but the switch stops processing new updates for 600 ms after every 2 s.

for hundreds of milliseconds (about 600 ms with 95% occupancy – Fig. 8) every 2 s. Unlike HP 5406zl, here the idleness frequency depends on time, not the number of updates. Decreasing the rate at which the controller issues updates does not affect the idleness duration or frequency. During the period when the switch does not update its rules, it still responds to control plane messages (*e.g.*, barriers), but does it slightly slower, as if it was busy. We believe, this behavior allows the switch to reoptimize its flow tables or perform other periodic computations. We are in the process of explaining the root cause with the vendor.

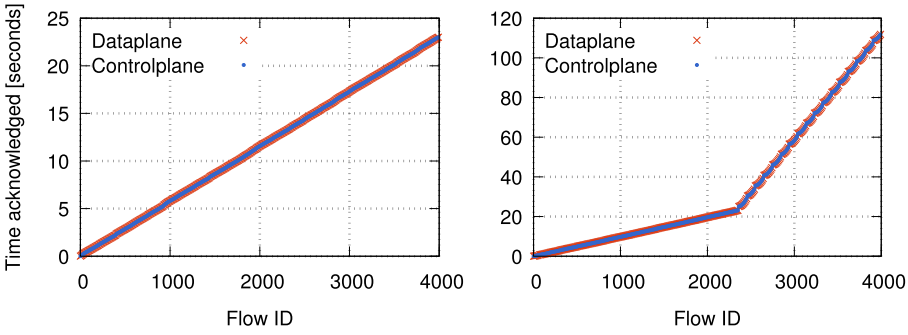
NoviSwitch 1132: Behavior reported in Fig. 2(f) repeats in longer experiments as well.

***Summary**: Flow table update rate often depends on the number of installed rules, but the control plane acknowledgments sometimes do not reflect this variability. A switch flow table state may be minutes behind what it reported to the control plane.*

4.3. Firmware updates can improve switch performance

We reported our findings to switch vendors and some of them provided us with new, improved firmware versions.

Switch X: Most notably, Switch X with firmware version V2, no longer allows for data and control plane desynchronization. As we show in Fig. 9, both views are synchronized and the rate does not increase when all control plane messages get processed, since they



(a) 7% (300 rules) (b) 50% (2250 rules)

Fig. 9. Control plane confirmation times and data plane probe results for the same flows in Switch X (firmware version V2). Data and control plane views are synchronized, but the rate still slows down after about 4600 flow installations.

are no longer processed before the data plane update ends. On the other hand, the switch still significantly slows down after about 4600 rule installations without full table cleaning. We repeat the experiment where we perform single rule installations and deletions, keeping flow table occupancy stable. Then, we stop an experiment and resume it after 10 min. Fig. 9(b) shows the results for occupancy of 50% (2250 rules). Behavior with the new firmware is the same as with the old version (Fig. 3(c)). Finally, at the beginning, the updates are slightly slower than in the previous version and slightly faster when the switch slows down (compare to Fig. 3).

NoviSwitch 1132: When we started our measurements of NoviSwitch 1132, the switch was running firmware version 250.3.2. The update rate was initially stable at about 60–70 rules/s, but after longer experiments started dropping to single digits and the switch required reboots. An investigation revealed that excessive logging was causing the disk space to run out in our longer and update-intensive experiments. We reported this fact to the vendor who provided us with a new firmware version: 250.4.4. A simple software upgrade allowed the switch to reach stable update rate of about 6000 rules/s — two orders of magnitude higher than before. Another upgrade (to version 300.0.1 used to get all measurements reported in this paper) increased the update rate by another 10–15% and fixed a bug that was causing the switch to crash when using long sequences of upgrades of rules with changing priorities.

***Summary**: Firmware is often responsible for switch faulty behavior and an upgrade can fix bugs or significantly improve performance without replacing hardware.*

4.4. Rule modifications are not atomic

Previously, we observed unexpected delays for rule insertions and deletions. A natural next step is to verify if modifying an existing rule exhibits a similar unexpected behavior.

A gap during a FlowMod: As before, we prepopulate the flow table with one low priority match-all rule dropping all packets and $R = 300$ flow specific rules forwarding packets to port α. Then, we modify these 300 rules to forward to port β. At the same time, we send data plane packets matching rules $101 - 200$ at a rate of about 1000 packets/s per flow. For each flow, we record a gap between when the last packet arrives at the interface connected to port α and when the first packet reaches an interface connected to β. Expected time difference is 1 ms because of our measurement precision, however, we observe gaps lasting up to 7.7, 12.4 and 190 ms on Pica8 P-3290, Dell 8132F and HP 5406zl respectively (Table 3). At HP 5406zl the longest gaps correspond to the switch inactivity times described earlier (flow 150, 200). A similar experiment with Switch X, Switch Y and NoviSwitch 1132 shows that average and maximum gaps are within our measurement precision.

Table 3
Time required to observe a change after a rule modification. The maximum time when packets do not reach either destination can be very long.

Switch	Pica8 P-3290	Dell 8132F	HP 5406zl
avg/max gap in packets [ms]	2.9/7.7	2.2/12.4	10/190

Drops: To investigate the forwarding gap issue further, we upgrade our experiment. First, we add a unique identifier to each packet, so that we can see if packets are being lost or reordered. Moreover, to get higher precision, we probe only a single rule (number 151 – a rule with an average gap, and number 150 – a rule with a long gap on HP 5406zl) and increase our probing rate to 5000 packets/s.

We observe that Pica8 P-3290 does not drop any packets. A continuous range of packets arrive at port α and the remaining packets at β. On the other hand, both Dell 8132F and HP 5406zl drop packets at the transition period for flow 150 (3 and 17 packets respectively). For flow number 150, HP 5406zl drops an unacceptable number of 782 packets. This suggests that the *update is not atomic*—a rule modification deactivates the old version and inserts the new one, with none of them forwarding packets during the transition.

Unexpected action: To validate the non-atomic modification hypothesis we propose two additional experiments. The setup is the same but in variant I the low priority rule forwards all traffic to port γ and in variant II, there is no low priority rule at all. Incorrectly, but as expected, in variant I both Dell 8132F and HP 5406zl forward packets in the transition period to port γ. The number and identifiers of packets captured on port γ fit exactly between the series captured at port α and β. Also unsurprisingly, in variant II, Dell 8132F floods the traffic during the transition to all ports (default behavior for this switch when there is no matching rule). What is unexpected is that HP 5406zl in variant II, instead of sending PacketIn messages to the controller (default when there is no matching rule), floods packets to all ports. We reported this finding to the HP 5406zl vendor and still wait for a response with a possible explanation of the root cause.

The only imperfection we observed at Pica8 P-3290 in this test is that if the modification changes the output port of the same rule between α and β frequently, some packets may arrive at the destination out of order. We did not record any issues with rule modifications in Switch Y and Switch X.

Finally we observed that NoviSwitch 1132 reorders packets belonging to different flows, but the timescale of this reordering (microseconds) is much below our probing frequency. That suggests,

Table 4
Combinations of overlapping low and high-priority rules.

Variant	R_{hi}		R_{lo}	
	IP src	IP dst	IP src	IP dst
I	exact	exact	exact	exact
II	exact	*	*	exact
III	*	exact	exact	*
IV	exact	exact	exact	*
V	*	exact	exact	exact

Table 5
Priority handling of overlapping rules. Both HP 5406zl and Pica8 P-3290 violate the OpenFlow specification.

Switch	Observed/inferred behavior
Switch X	OK
HP 5406zl	Ignores priority, last updated rule permanently wins
Pica8 P-3290	OK for the same match. For overlapping match may temporarily reorder (depending on wildcard combinations)
Dell 8132F	OK (Reorders within a batch)
Switch Y	OK
NoviSwitch 1132	OK

that the reordering is unrelated to an incorrect order of rule modifications. Indeed, we confirmed that packets in different flows get reordered even if there are no rule modifications. We also checked, that packets in the same flow do not get reordered. The switch vendor confirmed that packets belonging to different flows may be processed by different cores of the network processor. They also ensured us, that assuming not too complicated actions, the processing power should be sufficient even for small packets.

Summary *: Two out of six tested switches have a transition period during a rule modification when the network configuration is neither in the initial nor the final state. The observed action of forwarding packets to undesired ports is a security concern. Non-atomic flow modification contradicts the assumption made by controller developers and network update solutions. Our results suggest that either switches should be redesigned or the assumptions made by the controllers have to be revisited to guarantee network correctness.*

4.5. Priorities and overlapping rules

The OpenFlow specification clarifies that, if rules overlap (*i.e.*, two rules match the same packet), packets should always be processed only by the highest priority matching rule. Since our default setup with IP src/dst matches prevents rule overlapping, we run an additional experiment to verify the behavior of switches when rules overlap.

The idea of the experiment is to install (in the specified order) two different priority rules R_{hi} and R_{lo} that can match the same packet. R_{hi} has a higher priority and forwards traffic to port α, R_{lo} forwards traffic to port β. We test five variants of matches presented in Table 4. R_{hi} is always installed before and removed after R_{lo} to prevent packets from matching R_{lo}. Initially, there is one low priority drop-all rule and 150 pairs of R_{hi} and R_{lo}. Then we send 500 update batches, each removing and adding one rule: $(-R_{lo,1}, +R_{hi,151}), (-R_{hi,1}, +R_{lo,151}), (-R_{lo,2}, +R_{hi,152}), \ldots$ We send data plane traffic for 100 flows. If a switch works correctly, no packets should reach port β.

Table 5 summarizes the results. First, as we already noted, Dell 8132F, Switch Y, Switch X and NoviSwitch 1132 do not reorder updates between batches and therefore, there are no packets captured at port β in any variant. The only way to allow some packets on port β in Dell 8132F is to increase the batch size – the switch freely reorders updates inside a batch and seems to push them to hardware in order of priorities. On the other hand, Pica8 P-3290 applies updates in the correct order only if the high priority rule has the IP source specified. Otherwise, for a short period of time—210 ms on average, 410 ms maximum in the described experiment—packets follow the low priority rule. Our hypothesis is that the data structure used to store the software flow table sorts the rules such that when they are pushed to hardware the ones with IP source specified are pushed first. Finally, in HP 5406zl only the first few packets of each flow (for 80 ms on average, 103 ms max in this experiment) are forwarded to α and all the rest to β. We believe that the switch ignores the priorities in hardware (as hinted in documentation of the older firmware version) and treats rules installed later as more important. We confirm this hypothesis with additional experiments not reported here. Further, because the priorities are trimmed in hardware, when installing two rules with exactly the same match but different priorities and actions the switch returns an error.

Summary: *Results (Table 5) suggest that switches may permanently or temporarily forward according to incorrect, low priority rules.*

5. Flow table update speed

The goal of the next set of experiments is to pinpoint the most important aspects that affect rule update speed. We first pick various performance-related parameters: the number of in-flight commands, current flow table occupancy, size of request batches, used priorities, rule access patterns. Then we sample the whole space of these parameters to identify the ones that cause some variation. From the previous section we know that although the control plane information is imprecise, in a long run the error becomes negligible, because all switches except for Switch X synchronize the data and control plane views regularly. Therefore, we rely on barriers to measure update rates in long running experiments used in this section. Based on the results, we select a few experimental configurations which highlight most of our findings and present them in Table 6.

5.1. Two in-flight batches keep the switch busy

Setting the number of commands a controller should send to the switch before receiving any acknowledgments is an important decision when building a controller [12]. Underutilizing or overloading the switch with commands is undesired. Here, we explore whether there is a tradeoff between rule update rate and the servicing delay (time between sending a command and the switch applying it).

We use the default setup with $R = 300$ and $B = 2000$ batches of rule updates. The controller sends batch $i + k$ only when it receives a barrier reply for batch number i. We vary k and report the average rule update rate, which we compute as $2^*B/T$ where T is the time between sending the first batch and receiving a barrier reply for the last and 2 comes from the fact that each batch contains one add and one delete.

Fig. 10 shows the average update rate. The rule update rate with one outstanding batch is low as the switch is idle for at least a network RTT. However, even two in-flight batches are usually sufficient to saturate tested switches given our network latencies. Thus, we use 2 in-flight batches in all following experiments. Since the update rate for NoviSwitch 1132 is often an order of magnitude higher than other switches, we use plots with a split y axis.

Looking deeper into the results, we notice that with a changing number of in-flight batches HP 5406zl responds in an unexpected way. In Fig. 11 we plot the barrier reply arrival times normalized to the time when the first batch was sent for $R = 300$, $B = 50$ and a number of in-flight batches varying between 1 and 50. We show the results for only 4 values to improve readability. If

Table 6
Dimensions of experimental parameters we report in this section. Note, that we also run experiments for other combinations of parameters to verify the conclusions.

Experiment	In-flight batches	Batch size (del+add)	Initial rules R
In-flight batches	1–20	1+1	300
Flow table occupancy	2	1+1	50 to max for switch
Priorities	as in Flow table occupancy + a single low priority rule in the flow table		
Access patterns	2	1+1	50 to max for switch +priorities
Working set	as in Flow table occupancy, vary the number of rules that are not updated during the experiment		
Batch size	2	1+1 to 20+20	300

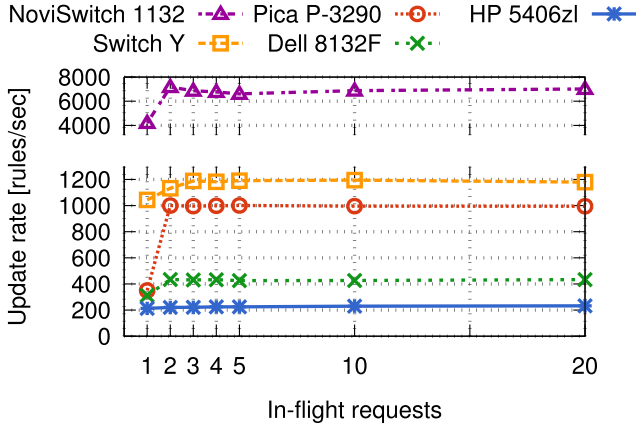


Fig. 10. Switch performance increases with the number of in-flight requests. However, the improvements beyond the case where the controller waits for confirmation of the previous request before sending the next one ($k = 1$) are negligible.

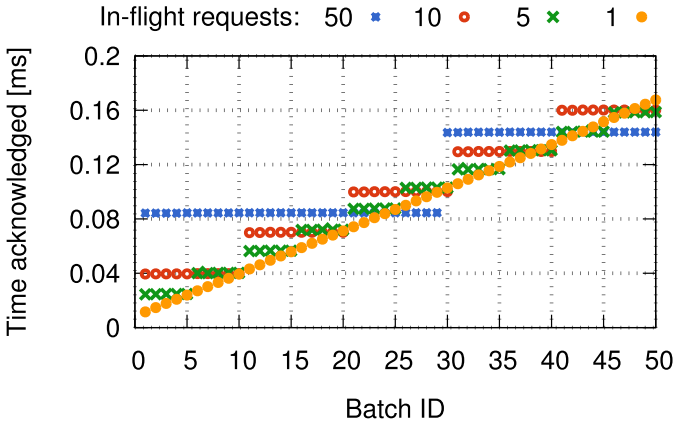


Fig. 11. HP 5406zl barrier reply arrival times. HP 5406zl postpones sending barrier replies until there are no more pending requests or there are 29 pending responses.

there are requests in the queue, the switch batches the responses and sends them together in bigger groups. If the constant stream of requests is shorter than 30, the switch waits to process all, otherwise, the first response comes after 29 requests. Moreover, the total processing time when the switch receives all updates at once is significantly shorter than for updates arriving in a closed loop. This can be due to the increased efficiency of batch processing and fewer context switches between message receiving and processing logic on the switch. The effect is visible, but much less pronounced for medium numbers of in-flight batches. This observation makes it difficult to build a controller that keeps the switch command queue short but full. The controller has to either let the queue get empty, or maintain the length longer than 30 batches. But based on the previous observation, even letting the queue to get empty has minimal impact on the throughput.

Summary: *We demonstrate that with LAN latencies two or three in-flight batches suffice to achieve full switch performance. Since, many in-flight requests increase the service time, controllers should send only a handful of requests at a time.*

5.2. Current flow table occupancy matters

The number of rules stored in a flow table is a very important parameter for a switch. Bigger tables allow for a fine grained traffic control. However, there is a well known tradeoff—TCAM space is expensive, so tables that allow complex matches usually have limited size.

We discover another, hidden cost of full flow tables. Namely, we analyze how the rule update rate is affected by the current number of rules installed in the flow table. We use the default setup fixing $B = 2000$ and changing the value of R.

In Fig. 12 we report the average rule update rate when varying switch flow table occupancy. There are three distinct patterns visible. Pica8 P-3290, Dell 8132F and Switch Y express similar behavior. The rule update rate is high when the flow table contains a small number of entries but quickly deteriorates as the number of entries increases. As we confirmed with one of the vendors and deduced based on statistics of another switch, there are two reasons why the performance drops when the number of rules in the table increases. First, even if a switch ultimately installs all rules in hardware, it keeps a software flow table as well. The flows are first updated in the software data structure which takes more time when the structure is bigger. Second, the rules need to be pushed into hardware (the switch ASIC), which may require rearranging the existing entries. Unlike other ASIC-based switches, HP 5406zl maintains a lower, but stable rate following a step function with a breaking point around 760 rules in the flow table. This stability is caused by periods of inactivity explained in Section 4. An update rate for NoviSwitch 1132 is an order of magnitude higher than for other switches. Additionally, the fast update rate (about 7000 updates/s) and its stability that is independent of the flow table occupancy for this device contrasts with all other switches.

Since Switch X update rate changes during an experiment and in older firmware version it does not offer a reliable way to measure its performance based on the control plane only, we manually computed 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 (Fig. 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

Table 7
Flow table update rate in Switch X depending on switch state and flow table occupancy. The rate gradually decreases with increasing number of rules in the flow table. After installing a total number of about 4600 rules, the switch update rate drastically decreases.

Occupancy	phase I	phase II	phase III
7% (300 rules)	415 rules/s	860 rules/s	–
25% (1125 rules)	374 rules/s	790 rules/s	34 rules/s
50% (2250 rules)	340 rules/s	–	28 rules/s
75% (3375 rules)	320 rules/s	–	20 rules/s
95% (4275 rules)	302 rules/s	–	8 rules/s

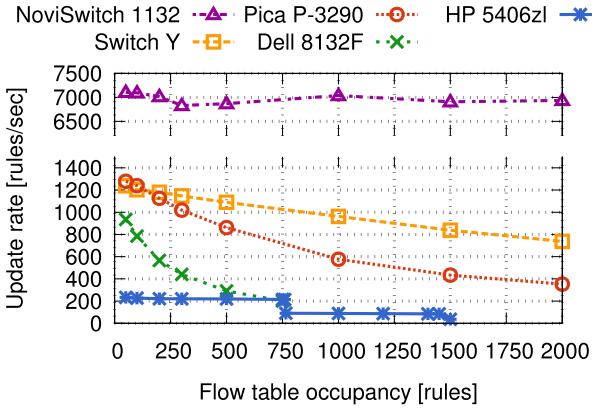


Fig. 12. For most switches the performance decreases when the number of rules in the flow table is higher.

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.

Fig. 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

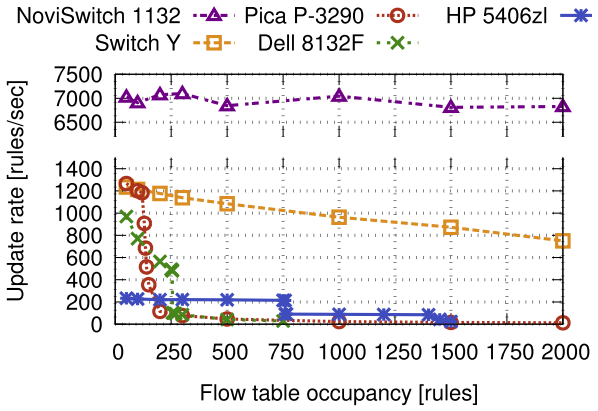


Fig. 13. Priorities cripple performance — Experiment from Fig. 12 repeated with a single additional low-priority rule installed reveals a massive fall in performance for two of the tested switches.

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.

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

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.

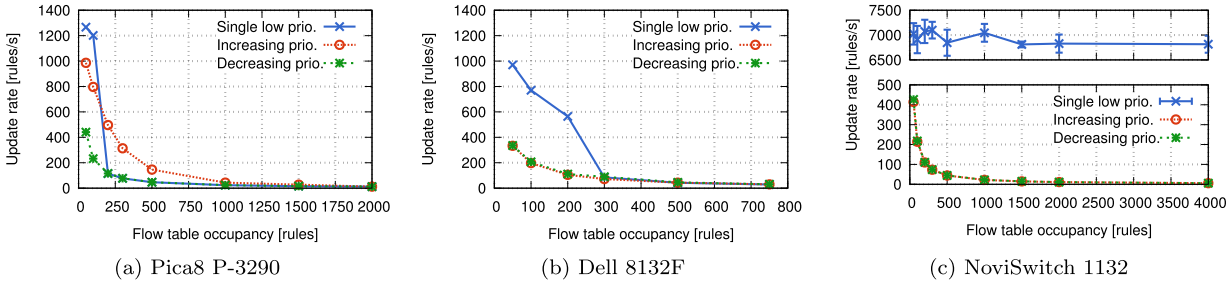
Finally, we rerun the experiments from Section 5.1 with a low priority rule. The rates for Pica8 P-3290 and Dell 8132F are lower, but the characteristics and the conclusions hold.

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 Fig. 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 performance is higher with only a single different priority rule until the breaking point, after which they become equal. Further, Pica8 P-3290 updates rules quicker in the increasing priority scenario.[9]

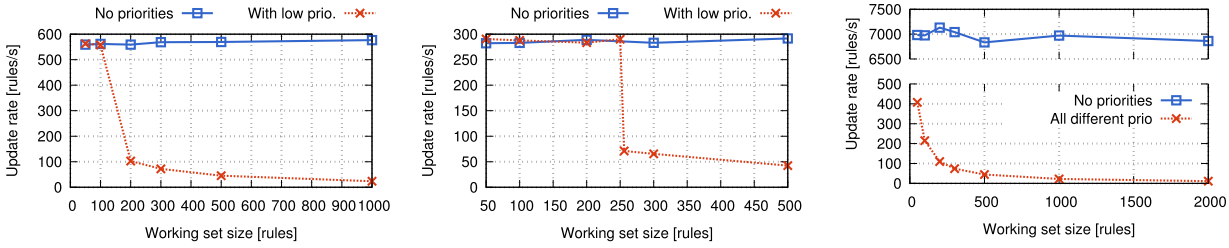
Fig. 14 shows that also NoviSwitch 1132 becomes significantly slower when there are additional priorities used as the update rate depends on the number of rules in the flow table. Even with just 50 installed rules, the rate drops from original 7000 updates/s to about 420. When the table occupancy increases 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 $D - \lfloor \frac{i}{10} \rfloor$ is almost equal to the rate with 100 initial rules in Fig. 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

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



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

Fig. 14. Switch rule update performance for different rule priority patterns.



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

Fig. 15. Size of the rule working set size affects 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).

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 affected 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 different priorities does not affect 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 unaffected and remains the same as presented in Figs. 12 and 13 both below and above the threshold of 760 rules in the flow table. Further, for both Pica8 P-3290 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 Fig. 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 (Fig. 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 Fig. 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 0.1-0.3 ms for Pica8 P-3290, 3.1–3.4 ms for Dell 8132F, 1 ms for Switch Y, 0.6-0.7 ms for HP 5406zl and 0.04 ms for NoviSwitch 1132. This explains the high overhead of Switch Y and Dell 8132F for fast rule installations in Fig. 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*

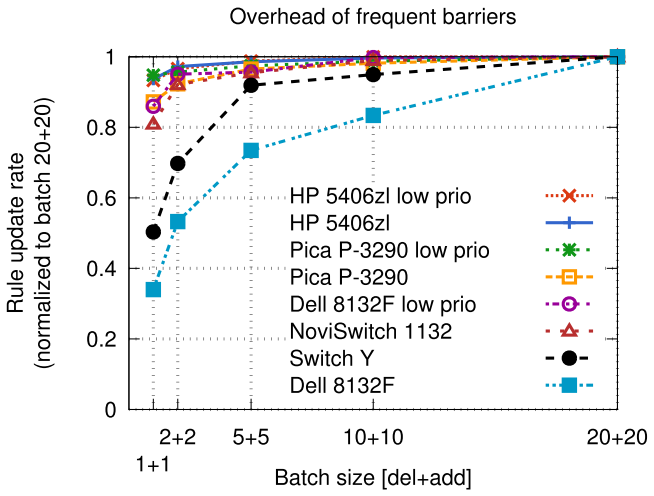


Fig. 16. Cost of frequent barriers is modest except for the case of Dell 8132F with no priorities (*i.e.*, with high baseline speed) and Switch Y where the cost is significant.

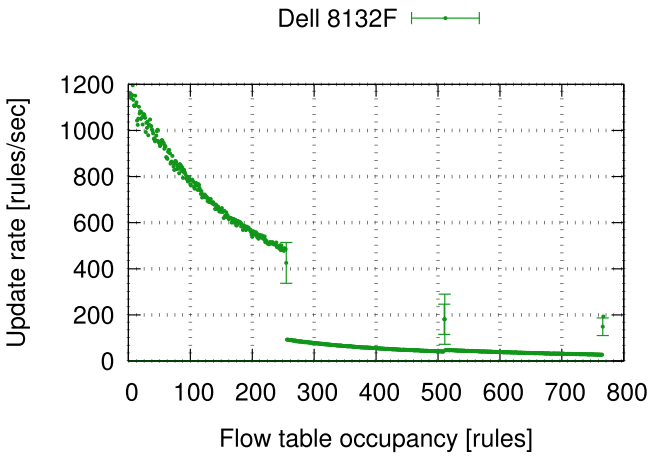


Fig. 17. An update rate in Dell 8132F suddenly increases for 4 specific flow table occupancy values.

balance between the switch performance and potential notification staleness. Moreover, there is a tradeoff 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 different 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 traffic can increase the update rate of Pica8 P-3290. We noticed that in some cases, sending data plane traffic 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 Fig. 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?, 2014, http://www.networkcomputing.com/networking/ethernet-switch-market-whos-winning/d/d-id/1234913.

[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, SIGCOMM, 2013.

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

[4] A. Greenberg, Microsoft showcases software defined networking innovation at SIGCOMM, 2015, https://azure.microsoft.com/en-us/blog/microsoft-showcases-software-defined-networking-innovation-at-sigcomm-v2/.

[5] Opendaylight, 2014, http://www.opendaylight.org/.

[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, HotSDN, 2013.

[9] R. Mahajan, R. Wattenhofer, On consistent updates in software defined networks, HotNets, 2013.

[10] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, D. Walker, Abstractions for network update, 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, SIGCOMM, 2013.

[12] P. Perešíni, M. Kuźniar, M. Canini, D. Kostić, ESPRES: transparent SDN update scheduling, 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, 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, HotSDN, 2013.

[16] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, Devoflow: scaling flow management for high-performance networks, 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? OSDI, 2010.

[18] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A.W. Moore, Oflops: an open framework for openflow switch evaluation, 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, CoNEXT, 2014.

[20] M. Kuźniar, P. Perešíni, D. Kostić, What you need to know about sdn flow tables, PAM, 2015.

[21] A. Khurshid, X. Zou, W. Zhou, M. Caesar, P.B. Godfrey, Veriflow: verifying network-wide invariants in real time, NSDI, 2013.

[22] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, S. Whyte, Real time network policy checking using header space analysis, NSDI, 2013.

[23] P. Kazemian, G. Varghese, N. McKeown, Header space analysis: static checking for networks, NSDI, 2012.

[24] N. Kang, Z. Liu, J. Rexford, D. Walker, Optimizing the "One Big Switch" abstraction in software-defined networks, CoNEXT, 2013.

[25] N. Katta, O. Alipourfard, J. Rexford, D. Walker, Cacheflow: dependency-aware rule-caching for software-defined networks, 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, SOSR, 2017.

[28] P. Perešíni, M. Kuźniar, D. Kostić, Openflow needs you! a call for a discussion about a cleaner openflow API, 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, 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 Netw. 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, 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, ONS, 2014.

[34] M. Yu, A. Wundsam, M. Raju, NOSIX: a lightweight portability layer for the SDN OS, ACM SIGCOMM Comput. Commun. Rev. 44 (2) (2014).

[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 affecting 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 different 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.