What you need to know about SDN control and data planes

EPFL Technical Report EPFL-REPORT-199497

Maciej Kuźniar[†], Peter Perešíni[†], Dejan Kostić[‡]
[†] EPFL
[‡]KTH Royal Institute of Technology
[†]<name.surname>@epfl.ch
[‡]dmk@kth.se

ABSTRACT

SDN and OpenFlow are actively being standardized and deployed. These deployments rely on switches that come from various vendors and differ in terms of their performance and available features. Understanding these differences and basic performance characteristics is essential for ensuring efficient deployments.

In this paper we measure, report, and explain the performance characteristics of the control- and data-planes in three hardware OpenFlow switches. Our results can help controller developers to make their programs 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. As a matter of fact OpenFlow deployments are now becoming a common occurrence both in datacenters [3] and wide-area networks [6]. This means that the number of SDN developers that are creating exciting new frameworks [1] as well as network administrators that are using a variety of SDN controllers is rapidly increasing[1]. An OpenFlow 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.

In OpenFlow, a logically centralized controller programs the network by manipulating forwarding rules in the OpenFlow switch flow tables. OpenFlow's transition from research to production means that the new frameworks are taking reliability and performance to new levels that are necessary in the production environment. For example, consistent network update schemes [8, 14, 17] 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 [13]. Moreover, approaches are appearing that are scheduling rule installations in an effort to minimize average rule installation time [14, 16]. All of these assume quick rule installation latency, and many rely on the so-called barrier command to gain positive acknowledgment from the switch's control plane before proceeding to the next step.

Initially, sporadic OpenFlow switch performance measurements were reported [4, 5, 19]. Recently, there has been early work on developing tests for OpenFlow switch implementations [18], and for example it has already shown that there are issues with the implementation of the barrier command.

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" which result in unexplained performance deviations from its usual behavior. Therefore, the thorough evaluation and explanation of 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. Same applies to trying out combinations of rule modifications.

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 Forwarding Information Base (FIB) update rate in hardware OpenFlow switches that support version 1.0 of this protocol. This paper is *not* about data plane forwarding performance. Our contributions are as follows: (*i*) we go a step further 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*) we believe we are the first ones to report several new types of anomalous behavior in OpenFlow switches.

Our key findings are as follows: (*i*) control plane performance is widely variable, and it depends on flow table size, priorities, size of batches and even rule update pat-

[1]We will refer to them all as SDN developers or developers for short.

Section	Key finding
4.1	Few outstanding requests are enough to saturate the switch.
4.2	Rule updates get slower as the flow table occupation gets higher.
4.3	Using rule priorities may degrade update performance by an order of magnitude.
4.4	Rule update patterns matter and switches can take advantage of an update locality.
4.5	Barriers are costly at some switches.
4.6	Rule modifications are slower than additions/deletions.
5.1	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.
5.2	Rule updates get reordered even if there is a barrier between them and they affect the same flows. Some switches ignore priorities.
5.3	Rule modification operation is non-atomic and switch may even flood packets for a transient period of time!

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

terns. In particular, priorities can cripple performance; (*ii*) switches might periodically or randomly stop processing control plane commands for up to 400 ms; (*iii*) data plane state might not reflect control plane—it might fall behind by up to 400 ms and it might also manifest rule installations in a different order; (*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.

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*. The existing tools that check if the control plane is correctly configured [9–11] 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 guarantees.

The benefits of our work are numerous. First, we hope that SDN controller and framework developers will find our findings useful in trying to ensure consistent performance and reliability from the variety of switches they might encounter. Thus, we report most of our findings with these developers in mind. For example, the existence of the performance anomalies underlies the difficulty of computing an off-line schedule for installation of 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 an ability to examine a large number of possible scenarios and pinpoint anomalies). Third, efforts that are modeling switch behavior [5], 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 may lay. If these problems still exist after five years of development, they may be caused by fundamental limitations that are hard or impossible to overcome, and could therefore be present in the future switch generations as well. An example of such a well known limitation, unrelated to performance though, is the flow table size. Researchers and switch developers understand that big TCAM is expensive and thus they try to save space in various ways [7].

The remainder of the paper is organized as follows. Section 2 presents background and related work, and we describe our measurement methodology in Section 3. We discuss in detail our findings about the control and data planes in Sections 4 and 5, respectively. We conclude in Section 6.

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 it as realized in 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. 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 forwarding table. The control plane at the switch is realized by an OpenFlow agent — firmware responsible for the communication with the controller and for applying the updates at the data plane.

The controller 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 acknowledgments that the update was performed [15]. The only way to deduce this information is to rely on the barrier command. As specified, after receiving a barrier request, the switch has to finish processing all previously-received messages before exe-

cuting any messages after the barrier request. When the processing is complete, the switch must send a barrier reply message [2].

Switch 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.* [19] report switch CPU-limited performance of about few hundred OpenFlow port status requests per second. Similarly, as part of their work on the Devoflow modifications of the OpenFlow model [4], Curtis *et al.* identify and explain the reasons for relatively slow rule installation rate on an HP OpenFlow switch. OFLOPS [18] is perhaps the first framework for OpenFlow switch evaluation. Its authors used it to perform fine-grained measurements of packet modification times, flow table update rate, and flow monitoring capabilities. This work made interesting observations, for example that some OpenFlow agents did not support the barrier command. OFLOPS also reported some delay between the control plane's rule installation and the data plane's ability to forward packets according to the new rule. Huang *et al.* [5] perform switch measurements while trying to build High-Fidelity Switch models that will be used during emulation with Open vSwitches. This work quantifies the variations in control path delays and the impact of flow table design (hardware, software, combinations thereof) at a coarse grain (average behavior). This paper also measures and reports surprisingly slow flow setup rates. Relative to these works, we dissect switch performance at a finer grain, over a longer period of time, and more systematically in terms of rule combinations, initial parameters, etc. In addition, we identify the thresholds that reveal previously unreported anomalous behavior.

Jive [12] proposes 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.

NOSIX [20] notices the diversity of OpenFlow switches and creates a layer of abstraction between the controller and the switches. This layer provides a translation of commands to optimize 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.

3. MEASUREMENT METHODOLOGY

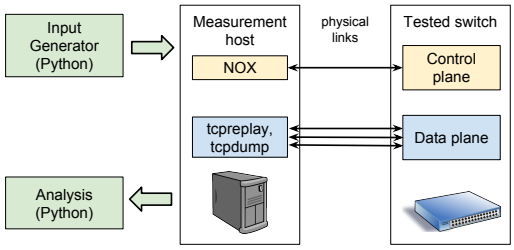


Figure 1: Overview of our measurement tools and testbed setup.

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

3.1 Tools and experimental setup

Based on our initial investigation, as well as on previously reported results [5], we make several observations. First, the switches we test are often in remote locations with limited physical access. Therefore, the measuring tool has to run on standard computers and cannot use customized hardware (*e.g.*, FPGAs). Moreover, our access is sometimes limited in time (*e.g.*, in Academic OpenFlow testbeds); we therefore expect the data collection and analysis to be separate. As the switches under test can typically modify at most a couple thousand of rules per second, we assume that a measurement precision in the order of a millisecond is sufficient. Finally, our previous experiments reveal that switches sometimes behave in an unexpected way and thus we need to tailor our experiments to locate and dissect the problem. Therefore, the main requirements for the measurement tool are (*i*) flexibility, (*ii*) portability, and (*iii*) sufficient precision.

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 (Figure 1).

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

The control plane measurement engine has to emulate the behavior of an OpenFlow controller (hence in the rest of the paper we sometimes refer to this component

Switch:	Pica8 P-3290	Dell 8132F	HP 5406zl
Flow table size:	∼2000	∼750	∼1500

Table 2: Approximate hardware flow table size for the tested switches.

as a controller). We choose to implement it as a module on top of the NOX controller platform. NOX is implemented in C++ and can issue rule updates at a rate that is much higher than what the hardware switches can handle.[2] 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 traffic on the controller interface with tcpdump.

Some of our experiments require injecting and recording data plane packets to precisely measure when the FIB is updated. We rely on existing tcpreplay and tcpdump tools to both send packets based on a pcap file and record them. To remove time synchronization issues, we follow a simple testbed setup with the switch connected to a single host by multiple links — 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 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 Python scripts.

3.2 Switches under test

We benchmark three switches with OpenFlow 1.0 support: HP ProCurve 5406zl with K.15.10.0009 firmware, Pica8 P-3290 with PicOS 2.0.4, and Dell PowerConnect 8132F with beta[3] OpenFlow support (both Pica8 P-3290 and Dell 8132F belong to the newest generation of OpenFlow switches). Such switches have two types of forwarding tables (also called flow tables): a software and a hardware table. The switches have various hardware flow table sizes (that we report in Table 2) as well as various levels of OpenFlow support. We make sure that all of them ultimately place the rules used in the tests in a hardware flow table. Moreover, while some switches implement a combined mode where packet forwarding is done by both hardware and software, we concentrate on the former, as the latter imposes high load on the switch's CPU and provides lower forwarding performance. Further, as mentioned before, analyzing the data plane forwarding performance is also out of scope of this paper.

[2] Our benchmark with software OpenVSwitch handles ∼42000 rule updates/s.

[3]The software is going to be optimized and productized in a near future.

Experiment	**In-flight batches**	**Batch size (del+add)**	**Initial rules R**
In-flight batches	1-20	1+1	300
Flow table size	2	1+1	50 to max for switch
Priorities	as in Flow table size + 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 size, vary the number of rules that are not updated during the experiment		
Batch size	2	1+1 to 20+20	300

Table 3: Dimensions of experimental parameters we report in this section. Note, that we also run experiments for other combinations of parameters to verify the conclusions.

Since each switch we test is located in a different institution, there are small differences between the testing machines and the network performance. However, the set-ups are comparable. A testing computer is always a server class machine and the network RTT varies between 0.1 and 0.5ms.

3.3 General experiment setup

In most experiments described in this paper we use the following generic setup and modify only particular parameters. At the beginning of each experiment we prepopulate the switch flow table with R default priority rules forwarding packets matching flows number 1 to R to port A. Then, we wait until the switch applies this update in the hardware flow table. Afterward the measured run starts. We send B batches of rule updates, each batch consisting of: B_D rule deletions, B_M rule modifications and B_A rule insertions followed by a barrier request. In the default setup $B_D = B_A = 1$ and $B_M = 0$. Moreover, unless specified otherwise, batch i deletes the rule matching flow number $i - R$ and installs a rule matching flow i with an action set to forwarding packets to port A. As a result, each batch removes the oldest rule. Note that the total number of rules in the table stays stable during the experiment.[4]

If the experiment requires injecting and capturing data plane traffic, we send packets that belong to flows F_{start} to F_{end} (inclusive) at a rate of about 1000 packets per flow per second (there is no exact number because the tool is limited by tcpreplay precision, which as explained before is sufficient).

For clarity, when describing an experiment we usually use only one variable, while keeping the other parameters constant. In reality we run the experiments with

[4] This is in a direct contrast to the previous work which considered only measuring the time it takes to fill a flow table starting from the empty initial state.

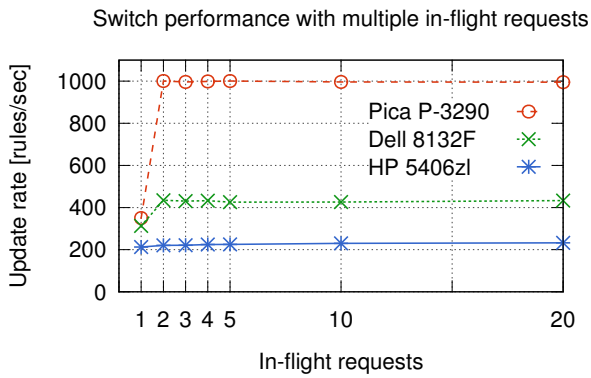


Figure 2: Switch performance increases with the number of in-flight requests. However, the improvements after the case where the controller waits for confirmation of the previous request before sending the next one ($k = 1$) are negligible.

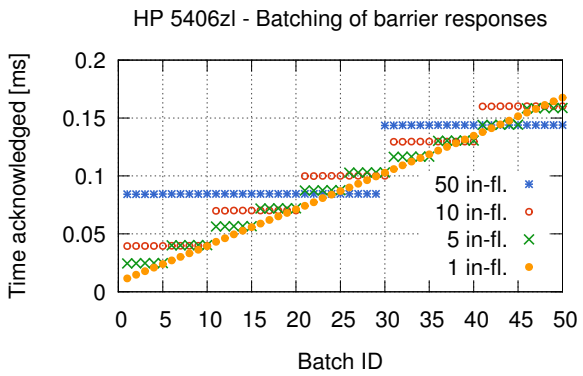


Figure 3: HP 5406zl barrier reply arrival times. HP 5406zl postpones sending barrier replies until there are no more pending requests or there are 29 pending responses.

different parameters as well to confirm the observations. Finally, we re-run each experiment at least three times and report the average of the measured results. Because the results have a small deviation across runs, unless otherwise specified, we do not show confidence intervals for clarity.

4. CONTROL PLANE PERFORMANCE

We first focus our experiments on the control plane interface. Our goal here is to pinpoint the most important aspects that affect the switch performance. We first identify various performance-related parameters – the number of in-flight commands, current flow table size, size of request batches, used priorities, rule access patterns, rule types, and action types. Then we sample the whole space of these parameters and try to identify the ones that cause some variation. We observe little impact caused by different match types, however, other parameters affect the performance. Based on the results, we select a few experimental configurations which highlight most of our findings in Table 3.

4.1 Two in-flight batches keep the switch busy

The number of commands a controller should send to the switch before receiving any acknowledgments is an important decision when building a controller [16]. Both underutilizing the switch and overloading it with commands is undesired. As sending a command is not instantaneous, the controller might want to send and queue several commands on the switch. As such, there is a tradeoff between achieving a high rule installation throughput (by keeping the switch always busy) and the "servicing" delay (the delay between the time the controller sends a command and the time the switch applies it). In the first experiment we quantify this tradeoff in order to find a performance sweet spot.

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.[5] We vary number k and report the average rule update rate which, because there is one add and one delete in each batch, we compute as $(1 + 1) * B$ divided by the time that passes between controller sending the first batch and receiving a barrier reply for the last batch.

Figure 2 shows the average rate across eight runs. Essentially, using a single outstanding batch means that the switch remains idle for at least a network RTT and the time it takes the controller to react to a barrier reply. Therefore, the rule update throughput is low. However, using only two outstanding batches is sufficient to saturate all tested switches given our network latencies and controller speed. Increasing the number of messages queued at the switch does not improve performance.

Looking deeper into the results, we notice that with a changing number of in-flight batches HP 5406zl responds in an unexpected way. In Figure 3 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. It is visible that, if 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 until it ends, otherwise, the first response comes after processing 29 requests. 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 through-

[5]Note that barrier replies come always in order, *i.e.*, we receive configuration of batch i only after all batches $< i$ are confirmed.

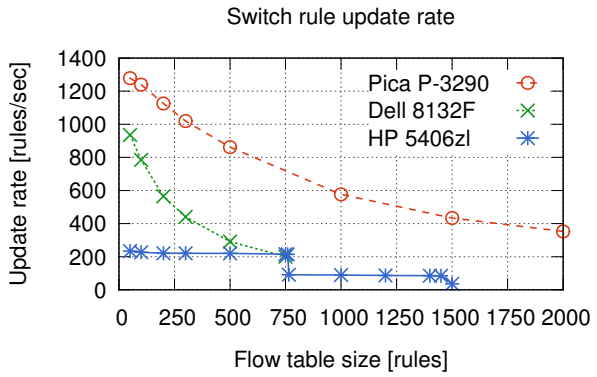


Figure 4: Switch performance decreases when the number of rules in the flow table is higher.

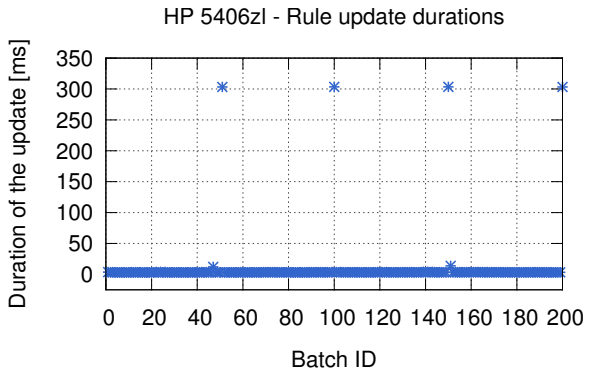


Figure 5: Time it took HP 5406zl to confirm batch installation. We can see that the switch periodically stops responding to the control plane for up to 300ms. We use a synchronous (single in-flight batch) setup in the experiment to rule out request queuing.

put.

***Summary**: In this experiment we demonstrate that with local area network latencies it is enough to use only two in-flight batches to achieve full switch performance. Moreover, with bigger number of in-flight requests, the service time increases as the requests are stuck in the switch command queue. Therefore, controllers should not send more than a handful of requests at a time.*

4.2 Current flow table size matters

The number of rules stored in a switch flow table is a very important parameter for a switch. A bigger flow table allows for a fine grained traffic control. However, there is a well known tradeoff—TCAM space is expensive, so the tables allowing complex matches have usually limited size (see Table 2). To work around the limited TCAM size, researchers propose solutions that balance rules across all available switches to avoid exceeding maximum flow table capacity [7].

We look at a similar problem from a different perspective. Namely, we analyze how the rule update rate is affected by the current number of rules installed in the flow table. Once again we use the default experimental setup fixing $B = 2000$ and changing the value of R. Based on the results of previous experiment, unless specified otherwise, here and in all following experiments we choose to use 2 in-flight batches.

In Figure 4 we report the average rule update rate over three runs. There are two distinct patterns visible. Both Pica8 P-3290 and Dell 8132F 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 the other 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 version and updating this data structure takes more time if the structure is bigger. Second, the rules need to be pushed into hardware—the switch ASIC—and this operation may require rearranging the existing entries.

On the other hand, HP 5406zl maintains a lower, but stable performance following a step function with a breaking point when there are around 760 rules installed in the flow table.

Moreover, the results presented so far report only an average rate over a long-running experiment. In practice, there is a high variance in a single batch processing time for both Pica8 P-3290 and HP 5406zl. Figure 5 shows the request duration for each batch—the time between the controller sends a batch and receives barrier reply for this batch. The flow table initially contains $R = 300$ rules and each batch consists of a rule deletion and insertion. Clearly, every 50 batches there is a 300ms long break when the switch is not acknowledging the previous barrier. This behavior is consistently reproducible. A further analysis with different batch types and initial rule numbers shows that for HP 5406zl:

- the frequency of long delays is related to the number of rule updates, rather than batches, and happens every 100 updates when the number of rules in the flow table is lower than 760.
- the frequency increases 3 times (to once every 33 updates) when the number of rules exceeds 760
- a modification of an existing rule counts as 2 updates.

This observation explains why the performance is so stable—it is dominated by these 300ms long periods without any response. Understanding this behavior is closely related to how the switch keeps the software and hardware flow tables synchronized and we postpone the discussion to Section 5.1 where we present additional data plane measurements to back up our reasoning.

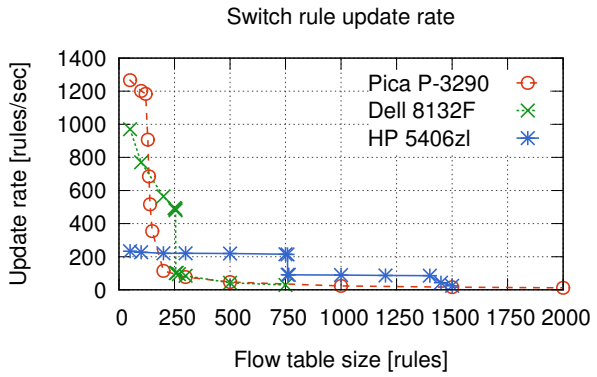


Figure 6: Priorities cripple performance — Experiment from Figure 4 repeated with a single additional low-priority rule installed reveals a massive fall in performance.

***Summary**: To sum up, the performance of all tested switches drops with an increasing number of installed rules, but the absolute values and the slope of this drop varies between switches. 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.*

4.3 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 always matches 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 4.2 with a single additional lowest priority drop-all rule installed at the beginning, before all flow-specific rules.

Figure 6 shows the rule update rates. If we compare it to previous results, we see that for a low flow table occupancy, all switches perform the same as without the low priority rule. However, both 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 does not suffer from the lowered rate so much, however, it also ignores the priorities (we investigate this issue further in Section 5.2). To make sure that the results are not affected by the fully wildcarded match in the low priority rule, we replace it by a specific IP source/destination match. The results remain the same and changing the drop action has no impact either.

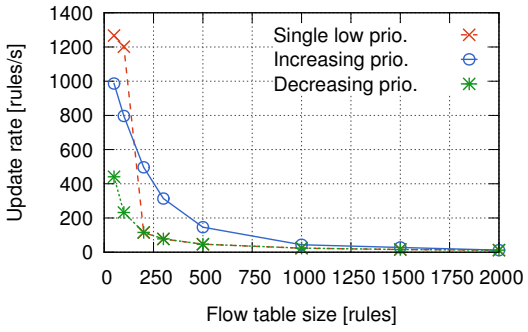
Taking these results into consideration we rerun the experiments presented in Section 4.1 with a low priority rule installed. As expected, the absolute rate is lower, but the characteristics and the conclusions hold.

***Summary**: This experiment demonstrates that the switch performance might be very difficult to predict. The presence of a single rule can degrade the FIB update performance of a switch by an order of magnitude. Controller developers should be aware of such behavior and avoid potential sources of inefficiencies.*

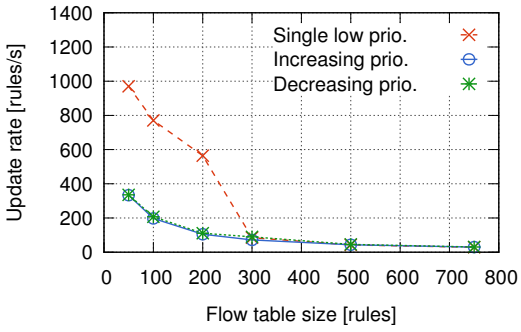
4.4 Rule access patterns matter

We look deeper in the results of the previous section and analyze how various update patterns and modification combinations affect the switch update performance.

More priorities: First, we check what is the effect of using different priorities for each rule. In this experiment 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.

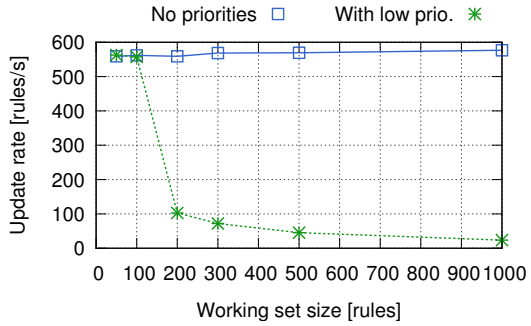


(a) Pica8 P-3290

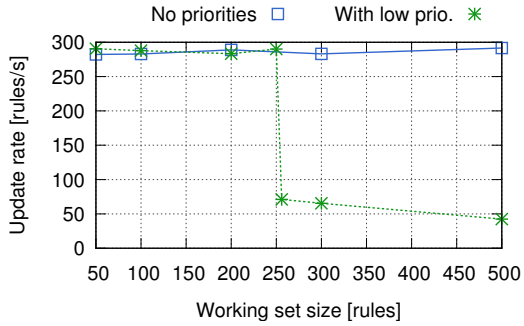


(b) Dell 8132F

Figure 7: Switch rule update performance for different rule access patterns.



(a) Pica8 P-3290



(b) Dell 8132F

Figure 8: 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).

Each tested switch reacts differently to such a workload. As it is visible in Figure 7, 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.[6] 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 5.4).

Working set size: Earlier we described the impact that the current number of rules has on the update rate. Here we analyze this problem further and check what happens if only a small subset of rules in the table (later referred as "working set") is frequently updated. We modify the default experiment setup such that batch i deletes the rule matching flow number $i - ws$ and installs a rule matching flow i. We vary the value of ws. In other words, assuming there are R rules initially in the flow table, the first $R - ws$ rules stay unchanged for the entire experiment and we update only the last ws rules.

The results show that HP 5406zl performance is unaffected and remains the same as presented in Figures 4 and 6 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. The performance is constant regardless of ws. However, when the low priority rule is installed, the update rate characteristic changes as shown in Figure 8. For both switches, as long as the update working set is smaller than their breaking point revealed in Section 4.2, 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 4.2 for table size ws.

[6] This is consistent with the observation made in [12], but the difference is smaller as for each addition we also delete the lowest priority rule.

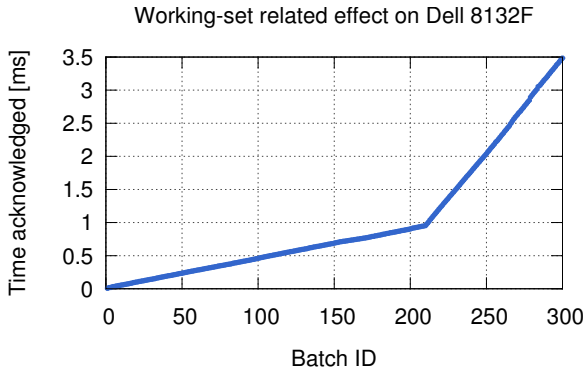


Figure 9: For a big working set and 300 initial rules, Dell 8132F significantly slows down at about 210 flows into the experiment. Afterward the rate stays constant.

Finally, working set related issues can be also spotted during our previous experiments with $ws = R$. In Figure 9 we show times when Dell 8132F acknowledged flows in control plane for $R = 300$. Most striking is the breaking point about 210 flows into the experiment, where the switch processing speed changes. Afterward, the switch keeps updating rules at a constant speed (verified up to 2000 batches). This breaking point is not present for low table occupancy, and varies depending on the number of rules for higher occupancy.

***Summary**: The bottom line is that some switches can optimize rule updates even if the table is full, but the updates affect only a small percentage of rules. While this information may be difficult to use in practice, it shows how difficult it is to predict and model switch behavior.*

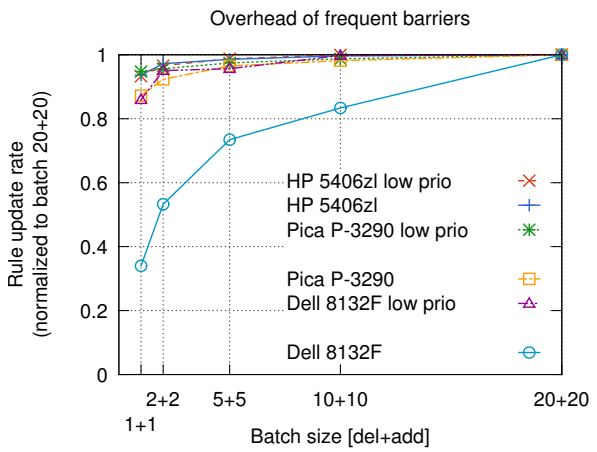


Figure 10: Cost of frequent barriers is modest except for the case of Dell 8132F with no priorities (*i.e.*, with high baseline speed) where the cost is significant.

4.5 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 10, 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.

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 and 0.6-0.7ms for HP 5406zl. This explains the very high overhead of Dell 8132F for fast rule installations in Figure 10 – barriers just take time comparable to rule installations. Finally, the reason of the unusually high barrier cost for Dell 8132F will become apparent later in our data plane measurements (Section 5.1).

***Summary**: Overall, we see that barrier cost can greatly vary across devices. The controller, therefore, should be aware of the potential impact and balance between the switch performance and potential notification staleness.*

4.6 Rule modifications are slower than additions and deletions

In addition to insertion and deletion, the third type of rule update is a modification of the existing rule. We run the same set of experiments as described in previous subsections, but using rule modifications instead. Because the results are very similar as in the previous experiments, we do not report them here in detail. All plots follow the same curves, but in general the update rate achieved for modifications is between 0.5x and 0.75x of the rate for additions and deletions for both Pica8 P-3290 and Dell 8132F. For HP 5406zl the difference is much smaller and stays within 12%.

5. DATA PLANE

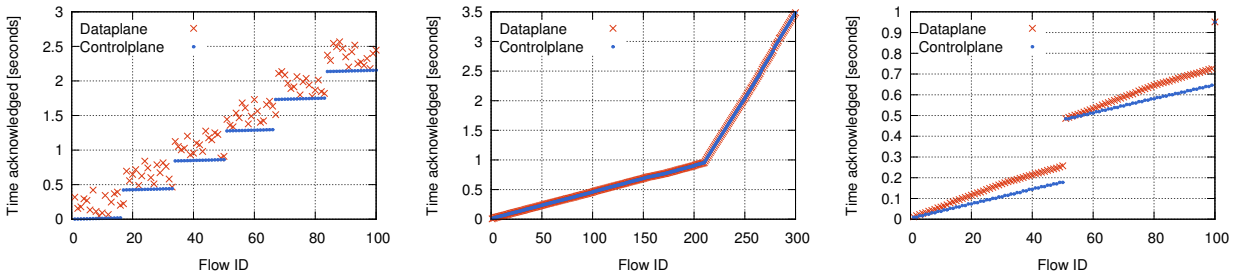
While the only view the controller has of the switch is through the control plane, the real traffic forwarding happens in the data plane. Therefore, it is important to understand how switches reflect control plane decisions in the data plane. In this section we present the results of experiments where we perform and monitor rule updates at the same time as sending and capturing traffic that exercises these rules. While control plane measurements highlight some differences between the switches, the data plane measurements show even bigger variability. Moreover, the unexpected behavior we observe may have negative implications for network security and controller correctness.

5.1 Synchronicity of control and data planes

Many solutions essential for correct and reliable OpenFlow deployments (*e.g.*, [13, 17]) 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. Therefore, it is crucial that this message works correctly. However, as authors of [18] already hinted, the switch view observed in the data plane may be different than the one advertised by the control plane. Taking this into consideration and worried by the variability of rule processing times observed in the control plane experiments we measure how do these two views correspond to each other at a fine granularity.

As before, we use a general setup with the match-all low priority rule that drops all packets[7] and R normal priority rules forwarding packets matching flows number 1 to R to port A. Then, we send B additional delete–install–barrier batches; batch i deletes the rule matching flow number $i - R$ and installs a rule match-

[7]This time, we are forced to use such a rule because otherwise the switch would flood the control channel with the PacketIn messages caused by data plane probes or flood the probes to all ports.



(a) Pica8 P-3290 (b) Dell 8132F (c) HP 5406zl

Figure 11: 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.

Switch	Data plane
Pica8 P-3290	reorders + behind up to 400ms
Dell 8132F	in sync with control plane
HP 5406zl	falls behind up to 250ms

Table 4: Data plane synchronicity key findings summary.

ing flow i with an action set to forwarding packets to port A.

At the same time, we inject data plane flows number F_{start} to F_{end} (inclusive). For each batch i we measure the time when a barrier reply for this batch arrives at the controller and when the first packet that belongs to flow i reaches the destination connected to port A. To work around the limited rate at which the testing machine can send and capture packets, we split the experiment in parts. In each part, we send traffic for a range of 100 flows. The results observed for HP 5406zl and Pica8 P-3290 are similar for each part. Therefore, we include the plot for only one range. For Dell 8132F we join together the results of three ranges to show the change in behavior after the breaking point mentioned in Section 4.4.

Figure 11 shows the results for $R = 300$, $B = 500$, $F_{start} = 301$ and $F_{end} = 400$ for HP 5406zl and Pica8 P-3290 and $F_{end} = 600$ for Dell 8132F. We do not present results for other values of the parameters as they are very similar. The first observation is that each switch behaves differently.

HP 5406zl: The data plane configuration of *HP 5406zl is slowly falling behind* the control plane acknowledgments – packets start reaching the destination long after the switch confirms the rule installation to the controller with a barrier reply. After about 100 rule updates (50 batches in the presented experiment), the *switch stops responding with barrier replies for 300ms*, which allows the data plane to catch up. After this time the data and control plane are synchronized and the process of diverging starts again. The divergence increases linearly and in this experiment reaches up to 82ms, but *can be as high as 250ms* depending on the number of rules in the flow table. The 300ms inactivity time stays constant across all the experiments we run, but happens three times more often if there are more than 760 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 300ms. 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, Pica8 P-3290 is either processing control plane (handling update commands and responding to barriers), or it is handling data plane (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 400ms* in this experiment. When all remaining rules get pushed into hardware, the switch starts accepting new commands in the control plane again. We contacted vendor of Pica8 P-3290 and got a confirmation that because the synchronization between the software and hardware flow 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, it's CPU is busy and therefore it stops dealing with the control plane.[8]

Dell 8132F: Finally, *Dell 8132F makes sure that no control plane confirmation is issued before a rule is actually installed* in hardware and becomes active. There are also no periods of idleness[9] as the switch pushes rules to hardware all the time and waits for completion

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

[9] 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. We expect them to be caused by periodic background processing the switch does.

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: Combinations of overlapping low and high-priority rules.

Switch	Observed/inferred behavior
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)
HP 5406zl	Ignores priority, last updated rule permanently wins

Table 6: Summary of priority handling of overlapping rules. Only Dell 8132F behaves as defined in the OpenFlow specification.

if necessary. We believe that this correct behavior justifies the fact that the barrier command on this switch has the biggest overhead. As already mentioned in Section 4.4, the switch starts updating rules quickly, but suddenly slows down after reaching the breaking point. However, even after the slowdown, the control plane reliably reflects the state of the data plane configuration.

Summary**: Placing rules in a hardware flow table is a costly process and to optimize it vendors often allow for temporary divergence between hardware and software flow table. Moreover, different vendors allow for different types (e.g., falling behind or reordering) and amounts of divergence (e.g., up to 400ms). Therefore, in general, a barrier command can not be used as a reliable method to guarantee flow installation. Moreover, to implement a correct controller, it is crucial to know what is the behavior of switches it is going to work with.* ***Ignoring the problem leads to an incorrect network state that may drop packets, or even worse, send them to an undesired destination!

5.2 Priorities and overlapping rules

Controllers often need to install rules that overlap (*e.g.*, two different rules that may match the same packet). OpenFlow specification clarifies that in such a case, only the rule with a higher priority should be matched (in case of equal priorities, the behavior is unspecified). In general, overlapping rules may result in data plane problems if (*i*) switch reorders rule installations, or (*ii*) priorities are not implemented properly. Finally, despite not being defined by the OpenFlow specification, reordering non-overlapping rules may be used as a technique to speed up the update speed without the risk of leading to network errors. On the other hand, overlapping rules should never be reordered if they are explicitly split by a barrier. Our previous experiments use specific IP source/destination matches that mean that rules do not overlap. We therefore run an additional experiment to verify the behavior of switches for overlapping rules.

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 A, R_{lo} forwards traffic to port B. We test five variants of matches presented in Table 5. The rules are always installed and removed in an order that does not allow packets to match R_{lo} ever — R_{hi} is always installed before and removed after R_{lo}. In this experiment there is initially one low priority drop-all rule and 150 pairs of R_{hi} and R_{lo}. Then we send 500 update batches. Each batch removes a single rule and installs a single rule, making sure that there is never a low priority rule installed unless the high priority corresponding one is there. We send data plane traffic for 100 flows. If a switch works correctly, no packets should arrive at port B.

The experiment results are summarized in Table 6. First, as we already observed, Dell 8132F does not reorder updates between batches and therefore, there are no packets captured at port B in any variant. The only way to allow some packets on port B 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—210ms on average, 410ms 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 80ms on average, 103ms max in this experiment) are forwarded to A and all the rest to B. That makes us believe that this 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, installing two rules with exactly the same match but different priorities and actions causes the switch to return an error.

***Summary**: Results of this experiment emphasize our previous point: despite a clear specification defining what barriers and priorities should mean, switch behavior often diverges from the ideal picture. These differences must be considered to achieve a correctly working net-*

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

Table 7: Time required to observe a change after a rule modification. The maximum time when packets do not reach either destination can be very long.

work.

5.3 Flow modifications are not atomic

In the previous experiments we observed unexpected delays in 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 A. Then, we modify these 300 rules to forward to port B. At the same time, we send data plane packets matching rules $101 - 200$ at a rate of about 1000 packets per second per flow. For each flow we measure when the last packet arrives at the interface connected to port A and when the first packet reaches an interface connected to B. In Table 7 we report the average and maximum delay between the two times. The first observation is that there is a noticeable gap between the packets stop being forwarded according to the old action and start using the new one. Further, for HP 5406zl there are some flows for which no packets arrive at any destination for almost 200ms — we checked that these flows correspond to the control plane response gap described earlier.

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 increase our probing rate to 5000 packets/s. However, we use only a single probing flow (number 150 – a flow with an average gap, and number 149 – a flow with a long gap on HP 5406zl). The initial setup and the update remain the same as before.

We observe that Pica8 P-3290 does not drop any packets. The packets belonging to a continuous range arrive at port A and then the remaining packets arrive at B. 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 149, HP 5406zl drops an unacceptable number of 782 packets. Thus we pose a hypothesis that the *update is not atomic*—a rule modification requires deactivating the old version and inserting the new one, with none of them forwarding packets at the transition time.

Unexpected action: To validate the non-atomic modification hypothesis we propose another two experiments. The setup is the same but in variant I the low priority rule forwards all traffic to port C and in variant II, there is no low priority rule at all. As expected, in variant I both Dell 8132F and HP 5406zl forward packets in the transition period to port C. The number and identifiers of packets captured on port C fit exactly between the series captured at port A and B. 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), the switch floods to all ports. We reported this finding to the HP 5406zl vendor and are currently waiting for their 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 A and B frequently, some packets may arrive at the destination out of order.

***Summary**: Even though occasional packet drops and reordering affects network performance, they are usually treated as acceptable in most networks. However, **forwarding traffic to undesired ports or even flooding them is a serious security concern**. Two out of three tested switches have a transition period during a rule modification. At this time the network configuration is neither the initial nor the final state. The fact that flow modification is not atomic stands in contradiction to the assumption usually made by controller developers and proposed solutions for network updates. We believe our results suggest that either switches should be redesigned or the assumptions made by the controllers have to be revisited to guarantee network correctness.*

5.4 Other surprises and trivia

In the process of running the experiments and digging deep to find and understand the root causes of various unexpected behaviors we made additional observations. They are not worth a section on their own because they have lower practical importance or we cannot fully explain and confirm them. However, we briefly report them as someone may find this information useful or inspiring to further investigate the issue.

HP 5406zl is very slow with priorities. We mentioned that HP 5406zl ignores priorities in the hardware and that in the presented experiments the data plane falls behind up to 250ms compared to the control plane acknowledgments. However, when different priorities are used, the switch becomes very slow in applying the changes in hardware without notifying the control plane. We run an experiment as in Section 4.3 but additionally install a low priority drop-all rule and inject data plane traffic. We observe that the hardware updates are consistently falling behind and the 300ms periods that the switch relies on to catch up are insuf-

ficient. Therefore, after the first 100 batches, *the delay reaches 7.5 seconds, and after another 300 batches it is as high as 30 seconds.*

Rule insertion may act as a modification. In one of the experiments we show that two out of three switches are unable to perform an atomic rule modification. However, when receiving a rule that has the same match and the same 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 actual rule modification. The behavior on Dell 8132F remains unchanged.

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. We are still investigating this issue and can not provide an explanation of this phenomenon nor confirm it with full certainty, but we report it anyway as something completely counter intuitive.

Dell 8132F performs well with a full flow table. In Section 4.3 we report 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 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.

6. CONCLUSIONS AND FUTURE WORK

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 three switches. 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 Marco Canini, 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 vendor for their quick response 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.

7. REFERENCES

[1] OpenDaylight. http://www.opendaylight.org/.
[2] OpenFlow Switch Specification. `http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf`.
[3] Ethernet Switch Market: Who's Winning?, 2014. http://www.networkcomputing.com/networking/ethernet-switch-market-whos-winning/d/d-id/1234913.
[4] A. Curtis, J. Mogul, J. Tourrilhes, and P. Yalagandula. DevoFlow: Scaling Flow Management for High-Performance Networks. In *SIGCOMM*, 2011.
[5] D. Y. Huang, K. Yocum, and A. C. Snoeren. High-fidelity switch models for software-defined network emulation. In *HotSDN*, 2013.
[6] 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, and A. Vahdat. B4: Experience with a Globally-Deployed Software Defined WAN. In *SIGCOMM*, 2013.
[7] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the "One Big Switch" Abstraction in Software-Defined Networks. In *CoNEXT*, 2013.
[8] N. P. Katta, J. Rexford, and D. Walker. Incremental Consistent Updates. In *HotSDN*, 2013.
[9] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real Time Network Policy Checking using Header Space Analysis. In *NSDI*, 2013.
[10] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In *NSDI*, 2012.
[11] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *NSDI*, 2013.
[12] A. Lazaris, D. Tahara, X. Huang, L. E. Li, A. Voellmy, Y. R. Yang, and M. Yu. Jive: Performance Driven Abstraction and Optimization for SDN. In *ONS*, 2014.
[13] H. H. Liu, X. Wu, M. Zhang, L. Yuan,

R. Wattenhofer, and D. A. Maltz. zUpdate : Updating Data Center Networks with Zero Loss. In *SIGCOMM*, 2013.

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

[15] P. Pereini, M. Kuzniar, and D. Kostic. OpenFlow Needs You! A Call for a Discussion about a Cleaner OpenFlow API. In *EWSDN*. IEEE, 2013.

[16] P. Perešíni, M. Kuźniar, M. Canini, and D. Kostić. ESPRES: Transparent SDN Update Scheduling. In *HotSDN*, 2014.

[17] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for Network Update. In *SIGCOMM*, 2012.

[18] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. Oflops: An open framework for openflow switch evaluation. In *PAM*, 2012.

[19] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the Production Network Be the Testbed? In *OSDI*, 2010.

[20] M. Yu, A. Wundsam, and M. Raju. NOSIX: A Lightweight Portability Layer for the SDN OS. *ACM SIGCOMM Computer Communication Review*, 44(2), 2014.