Performance Analysis of SDN Switches with Hardware and Software Flow Tables*

Piotr Rygielski
Institute of Computer Science
University of Würzburg,
Germany
piotr.rygielski@uni-wuerzburg.de

Marian Seliuchenko
Dept. of Telecommunication
Lviv Polytechnic National
University, Ukraine
m.seliuchenko@gmail.com

Samuel Kounev
Institute of Computer Science
University of Würzburg,
Germany
samuel.kounev@uni-wuerzburg.de

Mykhailo Klymash
Dept. of Telecommunication
Lviv Polytechnic National
University, Ukraine
mklimash@lp.edu.ua

ABSTRACT

Nowadays data centers are increasingly becoming larger and dynamic due to virtualization. Software-Defined Networking is the leading approach to network virtualization and flexible management. The wide variety of hardware implementations have brought strong heterogeneity to the market of networking devices which are different in terms of OpenFlow features and performance. In this paper we address the issue of heterogeneity of four hardware OpenFlow switches by characterizing selected performance relevant parameters for the hardware and software flow tables. We characterize maximum size of hardware flow tables for each switch including the behavior of a rule promotion engine that moves the rules between tables. We show that in the worst case forwarding packets using software table decreases the throughput by two orders of magnitude (from 940 to 14 Mbit/s). Our results can help the developers of SDN applications to account performance limitations of hardware and software processing as well as limited hardware support for a specific rule types.

CCS Concepts

•**Networks** → **Network performance analysis; Network measurement;**

Keywords

Software-defined networking; switching; flow tables; performance.

*This work was funded by the German Research Foundation (DFG) under grant No. KO 3445/18-1.

1. INTRODUCTION

Software-Defined Networking (SDN) has established a new standard for network virtualization by separating data plane from control plane and letting the user to develop custom software for the latter. OpenFlow [13] has become the standard protocol for communication between OpenFlow-enabled switches and SDN controllers. Many hardware vendors already offer OpenFlow-enabled network devices that support various versions of the OpenFlow protocol. The diversity of OpenFlow implementations (including both hardware capabilities and control plane software behaviors) makes understanding and control over a network difficult. As a result, the network performance becomes affected by the heterogeneity of OpenFlow switches and should be investigated to better understand the consequences of the design and configuration decisions of an SDN-based deployments.

In this paper, we focus on the influence of the heterogeneity of switch implementations on the performance of a software-defined network. We observe different behaviors and performance characteristics of OpenFlow-enabled devices. The main issue that motivates this work and concerns currently available devices is a change in the performance that depends on the state of the switch flow table [11]. Not all devices handle the rules in the flow tables in the same way; capacity of flow tables and rule compatibility differ among switch models. As shown in [10], different switches place the same rules differently as well as optimize the rules placement during the runtime. This can result in an unpredictable behavior where forwarding speed may drop drastically. The authors observed that most of hardware is still not mature enough for the performance to be predicted in the repeatable manner, because "each switch under test has many quirks which result in unexplained performance changes." They conclude that "the switch performance is difficult to predict—a single rule can degrade the update rate of a switch by an order of magnitude". The authors stress high diversity of the performance of the switches. On the other hand, the authors of [5] have shown that "The results also show that a more complicated SDN (...) does not necessarily mean performance degradation. Performance reflects the specific implementation of the SDN." These statements,

among others, are our main incentive for investigating the heterogeneity of the switches and its influence on the performance.

Our goal is to investigate the heterogeneity of four models of OpenFlow-enabled switches by the means of their: flow table types and capacities, behaviour of the rule insertion and promotion engines, flow tables compatibility with the various rule types and finally the switching performance. We manipulate the switch parameters and install forwarding rules into various flow tables to observe the performance of the switch. We aim to identify parameters that have strong influence on the performance. In particular, we answer the question about the performance of a switch when the hardware flow table is full. Additionally, we investigate what happens when some rules are removed from the tables and the rule placement can be reorganized. We focus on a proactive rule insertion, where controller installs rules before the matching packets arrive to the switch.

The main contributions of this paper are the following: (1) we investigate four models of physical switches and characterize their SDN capabilities, (2) we analyze the throughput performance of the switches in various SDN modes, and (3) we characterize the behavior of the rule management engines implemented in the switches.

This paper is organized as follows. In Section 2, we introduce the background of OpenFlow switch components that are relevant further in the paper. We present selected related work in Section 3. In Section 4, we show the testbed setup and describe how we measured the switches. In Section 5, we discuss the experiments and their results in details to finally conclude the paper in Section 6.

2. FOUNDATIONS OF SDN SWITCH PERFORMANCE

Software-defined networking assumes separation of the data plane and the control plane. In the data plane, a switch forwards the packets, whereas in the control plane, algorithms make decisions where the packets should be forwarded. The decisions are converted to a forwarding rules stored in the flow tables of the respective switches.

2.1 Hardware and Software Flow Tables

The rules saved in the switch can be exact-match (all match fields are specified) or wildcard-match (some fields have a value *any*). Exact match rules are stored in BCAM memory (binary content-addressable memory). The wildcard rules are saved in the TCAM (ternary content-addressable memory) that allows each cell to have three states: 1, 0, and $*$. Hardware flow tables based on TCAM usually have very limited capacity and capabilities because this memory is power hungry and expensive. The rules that do not fit to the *hardware flow tables* (either in BCAM or TCAM memory) are placed in the switch SDRAM—so called *software flow table*. As a rule of thumb, the hardware flow tables offer usually nearly the full line switching speed (do not inflict delay) but their volume is limited. We elaborate more on this in Section 5.

2.2 Reactive and Proactive Rule Insertion

If an arriving packet cannot be matched against any rule, the switch forwards the packet to the controller that reacts with inserting new rules. We call this a *reactive rule insertion*. In contrast, *proactive rule insertion* implies that the switches are preconfigured with the rules to handle all flows without reactive interaction with the controller. The *proactive rule insertion* fits better large scale data center scenarios where the paths usually stay constant, whereas the *reactive rule insertion* fits better smaller but highly dynamic networks (e.g., access networks or sensor networks [14]).

Although currently available SDN controllers do no scale well (as shown in, e.g., [14]), the authors of [12] point that the current commodity switches are unable to process all flows in hardware. For example, the authors of [2] point out that for large-scale delivery content networks there can be up to 16 million flows. This limits the performance of the reactive SDN on one hand (due to controller scalability issues) but also challenges proactive SDN setups on the other hand (as the capacity of the hardware flow tables is limited).

3. RELATED WORK

The topic of SDN performance attracts many researchers and some aspects have been already investigated in the literature. However, various authors usually focus on selected parts of SDN architecture so they miss the overall picture of the system (e.g., they focus on the performance of control path, data path, or a controller [10, 6, 3]), or model a network in a coarsely black-box manner [7]. Additionally, many authors (e.g., [5]) do not investigate physical switches but theorize their findings based on simulators or emulators. The relatively young concept of SDN resulted in multiple various hardware products that offer the support for the OpenFlow protocol. This variety resulted in different implementations and thus the offered performance can differ among the vendors or even among the switch models of the same vendor.

The authors of [9] reviewed almost 600 works concerning SDN and its aspects, among others the performance. They found out that "understanding the performance and limitation of the SDN concept is a requirement for its implementation in production networks. There are very few performance evaluation studies of OpenFlow and SDN architecture. Although simulation studies and experimentation are among the most widely used performance evaluation techniques, analytical modelling has its own benefits as well." In this Section, we extend their findings and propose an overview of the state-of-the-art in SDN performance.

As presented in [9], many authors use simulation techniques to solve performance models. In [8], the authors investigate various performance-related parameters of an SDN system. The system is modeled and examined in the OFSim simulator. The authors divide the process of packet forwarding into four stages: data plane, data-to-control plane, control plane, and control-to-data plane processing. They identify parameters correlated with the performance and examine the dependencies between achieved performance, table sizes and various message rates (*packetIn*, *flowmod*). The authors also note that "performance bottleneck may be located in the existing switches, and the flow table entry installation delay is a pressing issue." Unfortunately, they do not validate their simulator against a setup with physical switches, and support for data center networks are in their future plans. Moreover, the authors assume that the flow tables have size up to 20 thousand entries which is normally not the case for hardware forwarding tables, which deliver high performance. Using real hardware, the authors would

install the most of the rules in the software forwarding tables or their rules would be rejected by the switch.

The authors of [3] analyze the Linux implementation of the OpenFlow on a commodity server investigating the performance of the data plane. Their setup is similar to the NFV approach where the networking hardware is replaced by commodity servers. The analysis is conducted for single- and multi-flow case, so that the forwarding performance can be examined *with* (for single flow) and *without* the presence of the limited size of forwarding table (for multi-flow). Additionally the authors test the packet processing delays for exact-match and wildcard rules as well as for different types of flow tables and their sizes (usually 25-29μs). Although their results are relevant for a combination of SDN and NFV, they lack the context of a data center, where usually hardware network switches are used.

Similarly, the authors of [5] do not examine hardware switches to investigate the performance. They "indicate that SDN does have a performance penalty; however, it is not necessarily related to the complexity level of the underlying SDN infrastructure". In their paper, they compare performance of an SDN setup with performance of non-SDN networking devices. They use software SDN switch and Linux-based ProGFE which is an alternative to OpenFlow.

On the other hand, the authors of [6], investigate four various SDN switches (including the software OpenvSwitch) in a simple experimental testbed. Their aim is to design an emulator of an OpenFlow switch. They "focus primarily on vendor-specific variations in the control plane, although data planes can also be different across vendors." Their findings focus on the controller performance and thus are not applicable to proactive rule insertion that we focus on. They observe that network switch "emulators do not attempt to reproduce vendor-specific details of particular pieces of hardware, as end-to-end performance is typically dominated by implementation-agnostic features like routing (choice of links), link contention, and end-system protocols (application or transport-layer)." Moreover, they find out "that an appropriately calibrated emulation infrastructure can approximate the behaviour of the switches we study." Although their findings do not concern our point of view at the performance analysis directly (they focus mainly on control plane), their findings support our statement that a medium-grained end-to-end performance model of SDN-based data center can be built disregarding the heterogeneity of the switches.

On the other hand, there exist analytical performance models of OpenFlow switches. Jarschel et al. [7] model SDN using two queues (M/M/1-S and M/GI/I-S) and thus specify two processing paths having different performance—with and without the controller. The selection of the controller and non-controller path is modelled in a probabilistic manner. Unfortunately, the path over the software flow table that involves CPU processing on a switch is not modelled making the results applicable to an abstract SDN switch with unlimited TCAM capacity or to small use cases (cf. [12]).

Another analytical performance model was introduced by Azodolmolky et al. in [1]. The authors propose a model based on network calculus as "In contrast to queuing theory, network calculus is concerned with worst case (upper bounds) instead of average (equilibrium) behaviour and therefore does not deal with arrival and departure processes themselves but with bounding processes called arrival and service curves." [1]. They validate their model against the results obtained in the literature, namely in [15]. In [15], the authors propose a benchmark for OpenFlow switches and demonstrate it using three hardware switches of which one has full OpenFlow support and the rest run experimental firmware. The authors did not mention the models of the switches under test, however one of the two switches with incomplete OpenFlow support seems to use the software flow table (switching latency was reported as $\approx 300\mu s$). Unfortunately, in [1], the authors do not investigate the performance of the switching using software flow tables.

Many other authors, for example [11, 4] stress the diversity of switch capabilities and behaviors what makes network harder to understand and control. Current OpenFlow switch implementations may lead to performance bottlenecks with respect to the CPU load. Also the flow tables where the rules are inserted may be non-deterministic—on some switches rules may be rejected whereas on the others, rules may be installed into the software or hardware flow table.

It is important to stress that the understanding of the SDN performance with software and hardware flow tables is as important as understanding the behavior of the SDN controllers. Faced by the fact, that the flow tables are simply too small to accommodate enough flows for production scenarios, we investigate the performance and characteristics of four heterogeneous switches to provide more insight towards building a complete performance model. The currently available performance models usually investigate the performance of SDN controllers, but the forwarding performance using the software flow table is not addressed, although it cannot be neglected.

4. SWITCHES UNDER TEST AND EXPERIMENTAL ENVIRONMENT

In this paper, our ultimate goal is to investigate the behavior of hardware OpenFlow switches in various configurations, especially in the switching that uses the software flow table. In this Section, we present our testbed, its setup, and measurement methodology.

4.1 Switches Under Test

We evaluate the performance of four hardware OpenFlow switches. The exact models and their specifications are presented in Table 1. We investigate four heterogeneous HP switches. We characterize briefly the main differences between them. The models 2920 and 3500*yl* use *ProVision* operating system, whereas 5130 and 5700 use *Comware*. The switches based on *ProVision* system have two flow tables: one hardware and one software flow table. The 2920 supports only OpenFlow v1.0, whereas 3500*yl* supports OpenFlow *v1.0* and *v1.3*. The 2920 exposes both hardware and software table as a single abstract table that is reported by HP VAN SDN controller as table with *n/a*. Thus, the controller is not aware of multiple tables and cannot manipulate the rules location directly. The only possibility to distinguish the flow location is to query the switch operating system directly over console. The switch 3500*yl* acts identically to 2920 if configured to support OpenFlow *v1.0*. For OpenFlow *v1.3* it exposes each table with the preconfigured ID, unfortunately without any reference to the type of the table (hardware or software). The switches 5700 and 5130 support only OpenFlow *v1.3* and contain four hard-

Table 1: Switches under test: models and switching performance in native non-SDN mode (source: vendor data sheet).

Switch Model	Firmware Version	Switching Capacity	Switching Throughput
HP 2920-24G (J9726A)	$WB.15.12.0015$	92.2 $Mpps$	128.0 $Gbps$
HP 3500yl-24G (J8692A)	$K.15.17.0007$	75.7 $Mpps$	101.8 $Gbps$
HPE FF 5700-32XGT-8XG-2QSFP+ (JG898A)	$2422P01$	714.2 $Mpps$	960.0 $Gbps$
HPE 5130-24G-4SFP+ EI (JG932A)	$3111P03$	96.0 $Mpps$	128.0 $Gbps$

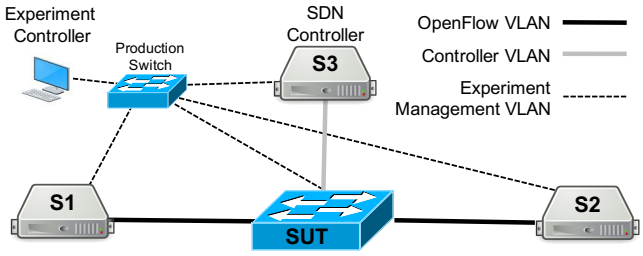

Figure 1: The overview of the experimental testbed.

ware flow tables which are: *extensibility*, *mac-ip*, *egress-vlan* and *ingress-vlan*. Both switches do not contain software flow tables. The ingress-vlan and egress-vlan tables provide support for tunnelling as defined by IEEE 802.1q, however, when any of these tables is in use, the connection to the controller is lost. The extensibility table supports all OpenFlow-mandatory matching fields, in contrast MAC-IP table supports only *vlan_vid*, *eth_dst* and *ip_dst*.

4.2 Experimental Testbed

Our experimental environment consists of the switch under test (SUT) and four servers that are connected to the production switch. The overview of the testbed is presented in Figure 1. The non-SDN production switch is used to control the experiment management VLAN. It connects all devices of the testbed using separate VLAN and separate physical cables so the OpenFlow network is isolated from the experiment management network. In this way, we can assure that the measured performance is not influenced by the experiment control traffic.

We use two servers ($S1$ and $S2$) to generate, receive, and measure the forwarding performance. The servers are connected to the SUT over a dedicated OpenFlow VLAN using the one gigabit copper interface and the Cat 6 cable. The third server $S3$ hosts HP SDN VAN Controller (version 2.5.15.1175) and is connected to the SUT over the controller VLAN. Finally, the fourth server—experiment controller—hosts the experiment control software that we developed to automate the experiments. The experiment controller connects to the generator and receiver servers over the SSH protocol and controls the traffic generation leveraging *iperf* and *ping* tools to measure SUT's throughput and the packet round-trip time. The experiment controller connects additionally to the SUT over *ssh* and polls hardware parameters (e.g., CPU load, the number of flows in the software and hardware tables, and the location of a given flow) that are not available over the controller. The experiment controller uses the REST API of the HP VAN SDN Controller to manipulate the flow tables of the SUT.

5. RESULTS

In this Section, we present the results of performance characterization measurements for the four switches under test. We focus on three distinguished aspects: hardware flow table capacities, forwarding performance and the behavior of the rule promotion engine.

5.1 Hardware Flow Table Capacities

The flow tables that handle the SDN forwarding rules have limited capacity. We identify this limit as the most important factor that influences the behavior of the switches. In this Section, we measure their maximal capacities with respect to the rule type.

5.1.1 Measurement Procedure

In order to measure maximal number of supported rules by a flow table we used standard 12 tuple matching structure with *output to port* action. For each switch we created set of rules supported by its hardware flow table. The rules are sorted by the number of wildcarded fields in descending order in every set. At the beginning we select the first rule form the respective set (i.e., having as many exact-match fields as possible). Then, we restart the OpenFlow instance at the switch to clear all flow tables and assure identical initial settings before every measurement. We use the direct ssh connection to the switch operating system to assure its clean state. Since the flow table contains rules that controller installs by default (e.g., redirecting packet to controller or to the software table), we pull the rules number and store this number as a reference. Next, we insert one rule and pull the rules number to verify if the rule was inserted properly. If the rule counter does not change then the table is full. We report the table capacity for the given rule structure. Next, we repeat this procedure for all rules in the respective set.

5.1.2 Capacities of Hardware Flow Tables

Results of the measurements for all switches are shown in Table 2. Table 2 should be read as follows. We assume that all fields are wildcarded, except the fields specified in the *match field* column. So, for example in row #6, the switch 3500 can store maximally 1526 rules having the following matching structure: fields *in_port*, *vlan_vid*, *vlan_pcp*, *eth_src*, *eth_dst*, *eth_type* are defined (except of *eth_src* and *eth_dst* that are required to be left empty), whereas the rest of the match fields are wildcarded. Zeros mean incompatibility with the given switch model.

Concerning the hardware limitations, the switch 2920 can store maximally 460 rules if all fields are wildcarded (except the *in_port* and *vlan_vid* that are not allowed to be wildcarded for this model). Considering the rest of the fields, any combination of those allows to create maximum of 460 unique rules in hardware. All rules that are not supported

Table 2: Hardware flow table capacities for various rule structures.

Switch		2920	3500	3500	5130	5130	5700	5700
OpenFlow Version		*v*1.0	*v*1.0	*v*1.3	*v*1.3	*v*1.3	*v*1.3	*v*1.3
Flow Table					*EXT*	*MAC-IP*	*EXT*	*MAC-IP*
Row	Match field							
#1	*in_port*	460	381	0	384	0	640	0
#2	*vlan_vid*	460	381	0	384	0	640	0
#3	*vlan_pcp*	460	0	0	384	0	640	0
#4	*eth_src*	0	0	0	384	0	640	0
#5	*eth_dst*	0	0	0	384	16000	640	65535
#6	*eth_type*	460	1526	1526	384	16000	512	65535
#7	*ip_dscp*	460	1526	1526	512	0	512	0
#8	*ip_src*	460	1526	1526	512	0	512	0
#9	*ip_dst*	460	1526	1526	512	16000	512	65535
#10	*ip_proto*	460	1526	1526	512	0	512	0
#11	*tcp_src*	460	1526	1526	512	0	512	0
#12	*tcp_dst*	460	1526	1526	512	0	512	0

in the hardware flow table are propagated to the software flow table. The capacity of the hardware table of the 3500*yl* switch depends on the OpenFlow version. If we set the version of OpenFlow to *v1.0* and the matching structure contains only *in_port*, the hardware flow table can store 381 rules with different priorities. This is also true for the *in_port* and *vlan_vid* fields together or just the *vlan_vid* alone. The maximum number of rules with any combination of the rest of the fields is 1526. In OpenFlow *v*1.3, the switch discards rules that contain at least one of the fields: *in_port*, *vlan_id*. For any other combination of fields, the hardware flow table can store maximally 1526 rules. The firmware of 5700 switch allows to configure the maximal size of extensibility table up to 65535 entries, however in practice the limitations are significantly lower. Maximal number of rules that do not include the field *eth_type*, IP fields and TCP fields is 512. In all other cases, the maximal rules number is 640. The *MAC-IP* table of the 5700 has maximum size of 65535 rules. The extensibility table of the *5130* can contain maximum 384 rules if all IP and TCP fields are wildcarded. Otherwise, the maximum equals 512. The *MAC-IP* tables in 5130 can hold up to 16000 rules.

5.1.3 Sharing Hardware Resources in Hybrid Switches

Both *ProVision* switches (i.e., 2920 and 3500*yl*) are hybrid and can process packets using OpenFlow and non-OpenFlow pipelines simultaneously. The switch summary reported by the HP SDN VAN Controller contains an option *hybrid mode*. If it is enabled, then the controller inserts an extra rule in the last table of the pipeline—so called *table-miss* rule that wildcards all match fields and has the lowest priority 0. This rule forwards packet to the standard pipeline if no match is found for the packet in any flow table. The *ProVision* switches share their hardware memory between various types of rules (e.g., OpenFlow, ACL, QoS). We investigated the behavior of the *rule policy engine* that reserves a given share of rules (expressed in percent) for the OpenFlow use (command: `openflow limit policy-engine-usage 100`).

The procedure for measuring the shared OpenFlow table capacity consists of the following steps: (1) disable the OpenFlow instance, (2) change the policy engine usage value, (3) enable the OpenFlow instance, and (4) fill up the table with rules that have the maximum number of exact match fields specified (not wildcarded). The results are presented in Figure 2. As expected, the dependency is linear. The OpenFlow policy engine usage can be set with granularity of 1%. When policy engine usage value equals 0, no resources are allocated to OpenFlow pipeline, i.e, no rules can be inserted. Value of policy engine usage can be changed only when the OpenFlow instance in the operating system of the switch is disabled, therefore it is not possible to scale flow table size during runtime.

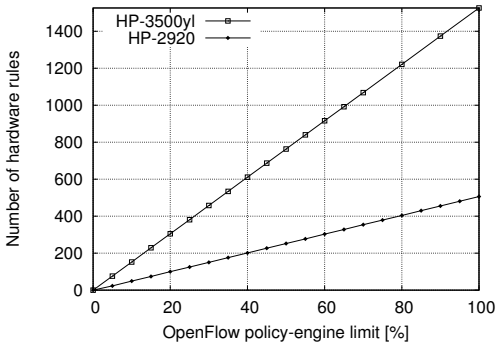

Figure 2: Maximal number of 8/2 wildcard rules (8 fields exact, 2 wildcarded) in the hardware table for varying OpenFlow share of the rule policy engine.

For the 3500*yl* switch, we observe a bug that results in the system failure when the policy engine value exceeds 60−65% and flow table size contains over 900 rules. Under these conditions, the OpenFlow instance of the switch hangs, packets are not forwarded, the connection to the controller is lost and any CLI command related to OpenFlow functionality freezes the console session. However, at the same time all other parts of the switch operate normally and CPU load is negligible low. The switch needs to be restated to function properly again.

5.2 SDN Forwarding Performance

In this section, we characterize the process of switching and describe our findings regarding the forwarding performance. Our observations concern mainly the differences

among the following two cases: forwarding using a rule placed in the hardware flow table and forwarding using a rule placed in the software flow table. In this paper, we do not investigate the forwarding performance with reactive rule insertion that involves sending *packet_in* message to the SDN controller and processing of the *flow_mod* reply. Instead, we focus on forwarding performance with proactive rule insertion, where rules are installed before the traffic arrives at the switch.

5.2.1 Measurement Procedure

We measure the maximal switching throughput and the CPU load for the switching using every flow table available in the given switch model. In this part, we generate traffic for 60 seconds using *iperf* and measure average throughput using the topology presented in Fig. 1. We make sure that the number of the installed rules is constant over the experiment, so that the *rule promotion engine*, which we describe in Section 5.3, does not affect the measurements.

5.2.2 Forwarding Performance for the Software and Hardware Flow Tables

The results for the forwarding throughput are presented in Figure 3. For the 2920 and 3500*yl*, we observe a drastic drop

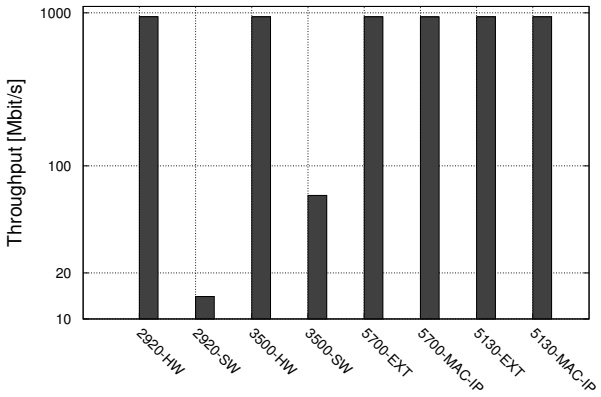

Figure 3: Maximal throughput obtained for the switches using various flow tables.

of the forwarding performance when a rule is installed in the software flow table (shortly denoted as SW in Fig. 3). The forwarding throughput of the hardware flow tables results in the full line-speed of the switch (about 941 Mbit/s) for all models, tables and OpenFlow versions. The performance in the non-OpenFlow mode is identical as when switching using the hardware OpenFlow table.

Forwarding using the software flow table causes the performance to drop from one (for the 3500*yl*) up to two orders of magnitude (for the 2920). A rule is placed in the software flow table (assuming it exists) if: (1) the rule is incompatible with the hardware flow table, or (2) the hardware flow table is full. This inflicts a serious constraint for the design of the software installed in the SDN controllers.

The switch forwarding capacity (expressed in packets per second) may be configured manually by changing *software-rate* parameter in the switch operating system. The switching capacity is expressed in packets per second. We changed the switching capacity using the *software rate* parameter (command: `openflow instance name limit software-rate <X>`) and observed the switch CPU load and the throughput. The maximum value of the *software rate* parameter is hard-coded in the switch and equals 2000 for the 2920 and 10000 for the 3500*yl*. In Figures 4 and 5, we present how the throughput and the switch CPU load depend on the limited switching capacity for the 2920 and 3500*yl* switches respectively.

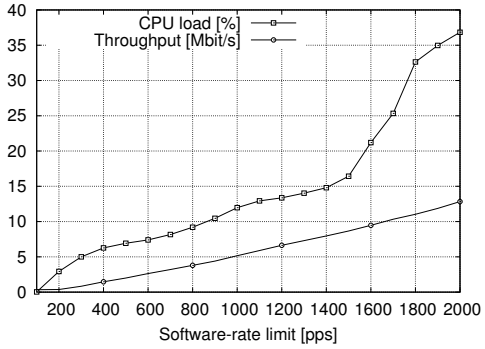

Figure 4: Throughput and switch CPU load for the software flow table of the switch 2920**.**

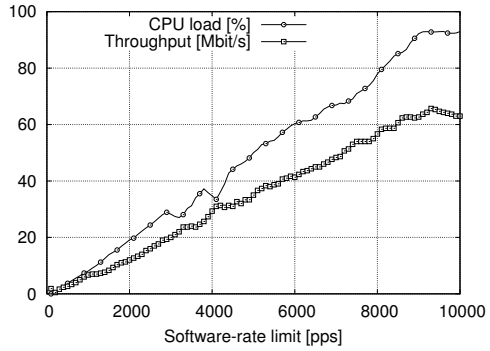

Figure 5: Throughput and switch CPU load for the software flow table of the switch 3500*yl***.**

Using the software flow table, the switches offer performance that practically eliminates them from the production environments. The 3500*yl* offers only about 6% of the maximum advertised throughput, whereas the 2920 provides less than 2%. This means that the developers of the SDN applications that control the behavior of an SDN-based network should be aware of the switch limitations and avoid filling the hardware flow table at all costs.

5.3 Rule Promotion Engine Behavior

Finally, we conduct experiments to investigate the behavior of the rule promotion and rule insertion engines. The *rule promotion engine* is responsible for moving the rules between the hardware and the software flow tables that are used simultaneously by the *ProVision*-based switches (3500*yl* and 2920). The *rule insertion engine* is responsible for selecting the flow table where a new rule should be placed. In this Section, we focus on the *rule promotion engine*.

5.3.1 Measurement Procedure

The behavior of the rule promotion engine has not been officially documented, so we set up an experiment to investigate its behavior in details. As the rule promotion engine operates only in the *ProVision*-based switches, we will experiment in this Section with the 2920 and 3500*yl* switches only.

In this experiment, we configure the 3500*yl* to operate in OpenFlow *v1.0*, since we observe, that for OpenFlow *v1.3* the switch does not move the rules between tables. For both switches, we set a software limit of the hardware table size to 76 for 3500*yl* and to 23 for 2920 by setting the *openflow policy-engine usage* to 5%. We want to install forwarding rules into the software flow table and then observe if the rules are promoted to the hardware flow table when occasion arises. Unfortunately, this is not possible since we cannot place the rules directly in the software flow table (the table does not have an *ID*). Thus, we first fill the hardware flow table with dummy rules (i.e., rules that can never be matched) and then install the two active rules that enable the *iperf* traffic in both directions between the experiment servers ($S1$ and $S2$ in Fig. 1). As a result, the *iperf* rules are installed in the software flow table as we aimed. We start to measure the forwarding throughput exactly as described in Section 5.2.

During the bandwidth measurement, the switch does not move the active rules between tables and the bandwidth stays low—none of the rules in the hardware flow table is actively matched. Then, we start removing the dummy rules from hardware table and the rules from the software flow table are being moved one by one to the hardware table as the space in the hardware flow table is freed. According to the OpenFlow specification, the less wildcarded rules should be matched first and should be placed at the beginning of any flow table. Therefore, we expect the switch to promote the rules with more exact-match structure in the first turn.

5.3.2 Prioritizing of the Rule Promotion

To determine the order in which the rules are promoted from the software table to the hardware one, we filled completely the hardware flow table of the 3500*yl* switch with dummy rules (76 rules) and inserted the same number of dummy rules in the software flow table.

For each dummy rule, we set the *timeout* parameter value calculated according to the following formula: *timeout=time offset+ order number of the rule*. The *time offset* prevents the switch from removing the rules from the table until we finish inserting all the rules necessary for the experiment, i.e., the parameter defines when the action of the rule promotion engine may be triggered. In the last step, we insert the *iperf* rules and start the throughput test. The *iperf* rules are more exact than all dummy rules. Every second during the measurement, we poll the switch for the current number of rules in the software and hardware flow tables. Additionally, we record the load level of switch CPU.

As shown in Figure 6, we observe that from the $\approx$ 36-th second, the 3500*yl* starts removing the rules from the hardware flow table (due to the timeout set to 30*s* and about 6*s* warm-up time) and replaces them with rules from the software flow table. At the $\approx$ 65-th second, both *iperf* rules are promoted to the hardware flow table and the CPU load drops down to 2%.

At the same moment—see Figure 7—the throughput of the 3500*yl* increased from 64 to 940 Mbit/s. Even though the *iperf* rules are the most exact, they were moved to the hardware flow table in the last turn. Therefore, the rules are moved from the software table to the hardware table in the reverse order than they were inserted. This statement is always true and does not depend on how many wildcarded fields the rules contain or what priority they have. This may lead to many performance drawbacks—e.g., the rules cannot be ordered for the promotion from the hardware flow table to the software flow table, because the rule promotion engine works in a first-come-first-served fashion. The analogous results for the 2920 are illustrated in Figure 8.

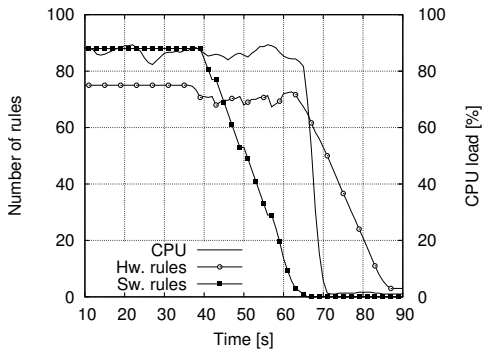

Figure 6: Behavior of the rule promotion engine for the 3500*yl*: numbers of rules in the flow tables. This Fig. has the x axis aligned with the Fig. 7.

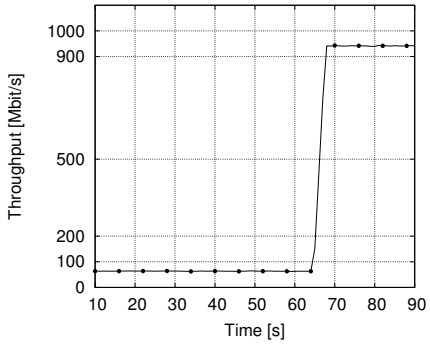

Figure 7: Behavior of the rule promotion engine for the 3500*yl*: throughput. This Fig. has the x axis aligned with the Fig. 6.

The experiments with the switch 2920 show that after the *iperf* rules are replaced from software to hardware, the throughput drops to 0 and the connection between the experiment servers is lost. The port counters in the switch detect arriving packets, however the packets are not forwarded between ports and no notification is sent to the controller. To restore the communication it is necessary to send a new *flow_mod* message and install the rules again.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have investigated the parameters that influence the performance of four heterogeneous OpenFlow-

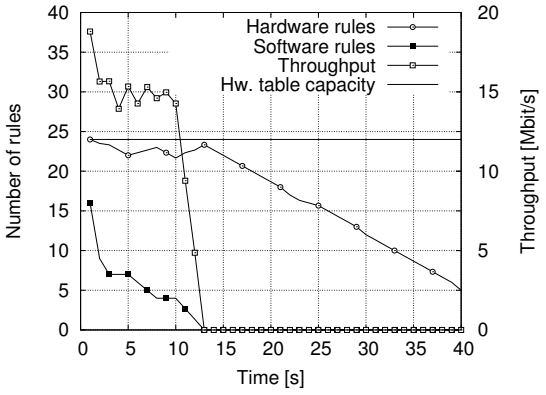

Figure 8: Behavior of the rule promotion engine for the 2920**: throughput**

enabled switches. In particular, we focused on capacities and forwarding performance of the hardware and software flow tables. We observed that the forwarding performance depends mainly on the location of the active rule. If the active rule is located in the software flow table, the performance is degraded by up to two orders of magnitude (941 vs. 14Mbit/s). The knowledge about the location of the active rule is thus crucial for the performance and should be acquired by every developer of the SDN controller applications. The location of a rule is influenced by many parameters such as: capacities of the TCAM and BCAM memories, behaviors of the rule promotion and insertion engines, and the compatibility of a given rule with the TCAM memory used in a given model of switch. This stresses the heterogeneity of OpenFlow-enabled switches—even if they are delivered by the same vendor. As a part of our future work, we plan to build a descriptive model that supports the modeling of the performance heterogeneity of SDN switches including their software and hardware flow tables.

7. REFERENCES

[1] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou. An analytical model for software defined networking: A network calculus-based approach. In *Global Communications Conference (GLOBECOM 2013)*, pages 1397–1402. IEEE, Dec 2013.

[2] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

[3] A. Bianco, R. Birke, L. Giraudo, and M. Palacin. OpenFlow Switching: Data Plane Performance. In *IEEE International Conference on Communications (ICC 2010)*, pages 1–5, May 2010.

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, pages 254–265. ACM, 2011.

[5] A. Gelberger, N. Yemini, and R. Giladi. Performance Analysis of Software-Defined Networking (SDN). In *IEEE 21st International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 389–393, Aug 2013.

[6] D. Y. Huang, K. Yocum, and A. C. Snoeren. High-fidelity switch models for software-defined network emulation. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 43–48, New York, NY, USA, 2013. ACM.

[7] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia. Modeling and Performance Evaluation of an OpenFlow Architecture. In *23rd International Teletraffic Congress (ITC 2011)*, San Francisco, CA, USA, 2011.

[8] X. Kong, Z. Wang, X. Shi, X. Yin, and D. Li. Performance evaluation of software-defined networking with real-life isp traffic. In *Computers and Communications (ISCC), 2013 IEEE Symposium on*, pages 000541–000547, July 2013.

[9] D. Kreutz, F. M. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[10] M. Kuzniar, P. Peresíni, and D. Kostic. What You Need to Know About SDN Flow Tables. In *Proceedings of the 16th International Conference on Passive and Active Measurement*, pages 347–359, 2015.

[11] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, Y. R. Yang, and M. Yu. Tango: Simplifying SDN Control with Automatic Switch Property Inference, Abstraction, and Optimization. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 199–212, New York, NY, USA, 2014. ACM.

[12] D. S. Marcon and M. P. Barcellos. Predictor: Providing fine-grained management and predictability in multi-tenant datacenter networks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 71–79, May 2015.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[14] R. Pries, M. Jarschel, and S. Goll. On the usability of openflow in data center environments. In *2012 IEEE International Conference on Communications (ICC)*, pages 5533–5537, June 2012.

[15] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. Oflops: An open framework for openflow switch evaluation. In *Proceedings of the 13th International Conference on Passive and Active Measurement (PAM 2012)*, pages 85–95, Berlin, Heidelberg, 2012. Springer.