It's About Time: Analyzing Flow Table Update Latency in SDN Switch Architectures

Fabricio M. Mazzola[1], Daniel S. Marcon[1,2], Miguel C. Neves[1], and Marinho P. Barcellos[1]

[1]Institute of Informatics – Federal University of Rio Grande do Sul, RS, Brazil
[2]University of Vale do Rio dos Sinos - Unisinos, RS, Brazil
Email: {fmmazzola, daniel.stefani, mcneves, marinho}@inf.ufrgs.br

***Abstract*—Forwarding devices are a key element of Software-Defined Networking (SDN). The architectures of these devices (also known as switches) differ with respect to implementation designs (software vs. hardware), use of memories (TCAM vs. RAM) and match-action structures (hash table vs. binary tree). These factors must be taken into account while predicting flow table update latency, i.e., how efficiently a switch will deal with control plane requests. Recent work either focuses on the performance of devices equipped with TCAM or performs a shallow comparison between a few different designs. In this paper, we perform an in-depth evaluation, highlighting differences among distinct switch architectures. Specifically, we define a robust measurement methodology and use it to determine the performance of switches when executing each key flow table operation (namely insertion, modification, and deletion) in multiple scenarios. Results show that different switch architectures and OpenFlow parameters can significantly influence flow table update latency. In particular, (i) the contrast in latency may scale up to three orders of magnitude for rule insertion; (ii) improper OpenFlow parameterization may increase flow setup times by a factor of 12x for modification and up to 6x for deletion within the same switch; and (*iii*) Open vSwitch presents variable performance and bimodal patterns for rule deletion. Our findings highlight the importance of understanding the operation of switches with different architectures and the need for an accurate configuration of the control plane.**

I. Introduction

Generalized forwarding introduced by Software-Defined Networking (SDN) [1] allows fine-grained management of flows. A logically centralized network controller installs rules (which describe network policies) in flow tables of forwarding devices (switches). These rules define how flows are handled by switches. Since the size of tables is typically limited, only a small number of rules can be placed in each switch at the same time. Given this constraint, switch operation usually requires rules to be inserted, modified and deleted from tables at potentially high frequencies. For example, in large-scale datacenter networks, millions of new active flows may arrive and need to be configured per second [2], [3]. A delayed flow table configuration may not only cause packets to be dropped, but also result in increased latency, negatively affecting time-sensitive applications (in this context, even 1 ms of added delay may be unacceptable [4], [5]).

Current forwarding devices are not limited to hardware designs using TCAMs (Ternary Content-Addressable Memory). Different trends (such as network virtualization [6] and programmable data planes [7]) have introduced new techniques to build SDN switches with different implementation designs (software vs. hardware), use of memories (TCAM vs. RAM) and match-action structures (hash table vs. binary tree). Additionally, each design may have certain details intrinsic to its implementation, such as limited support for rule priorities or types of matches (e.g., wildcard vs. exact match rules).

Given this heterogeneity, it is challenging to predict how these switches will perform when dealing with control plane requests. Previous work [8], [9], [10], [11] has found variability in performance when different switches process and handle control plane messages. Despite the contribution of previous studies, they are either limited by the depth of the comparison between switch designs or ignore altogether hybrid devices with different kinds of memory. Besides, as we will show, to date the differences in latency among architectures are not well understood.

In this paper, we conduct a comprehensive measurement of flow table update latency, highlighting differences among distinct switch architectures: hardware switch using TCAM, software switch using RAM, and a whitebox device with flow tables implemented as either TCAM or as RAM hash tables[1]. To evaluate these devices, we define a robust and systematic measurement methodology composed of multiple scenarios. More specifically, we use one similar instance of each switch architecture to measure the time to execute the three fundamental table update operations (rule insertion, modification and deletion). This latency reflects the time to modify the switch table only, disregarding propagation delay and system call latency. Moreover, we vary several factors in our experiments, such as rule match type, rule priority and OpenFlow parameters.

The main findings are described as follows. First, we show that latency disparity between switch architectures for rule insertion can be highly variable and reach up to three orders of magnitude, depending on the priority pattern used. Second, latency for rule modification and deletion is not influenced by rule match type and priority pattern. Surprisingly, it is significantly affected by the OpenFlow *strict* parameter, which, when absent, may increase flow configuration time within the same device up to 10x and 6x for rule modification and deletion, respectively. Finally, we reveal previously unknown anomalous behaviors, with high variability and bimodal patterns, for the

[1]Due to a limitation of the whitebox device, it was not possible to evaluate the operational mode combining TCAM and RAM tables simultaneously.

software switch (Open vSwitch – OVS) when removing flow entries.

Our observations show three crucial aspects for performance divergence in existing switch architectures. First, switch hardware is directly related to the efficiency of some table update operations, such as insertion with priority patterns. The intrinsic differences in flow table implementation among all architectures is highly responsible for the significant disparity of flow table update latency. The high latency may hinder time-sensitive applications, increase packet loss and decrease the efficiency of high-performance networks. Second, switch design decisions play an important role regarding the efficiency of table update operations. As we show in Section V, the high variability and bimodal behavior of OVS are caused by an implementation decision which negatively affects flow configuration time. Third, an inadequate use of OpenFlow parameters may significantly impact switch performance and consistency of network policies. These conclusions show that a low-level (architectural) understanding of how a switch operates and the correct use of OpenFlow parameters can drastically influence network performance.

The remainder of the paper is organized as follows. Section II examines related work and highlights the novelty of our paper. Section III provides a background on different existing switch architectures, detailing the set of devices used in this work and their main characteristics. In Section IV, we describe the methodology used to evaluate the switches accurately, including environment, metrics and general experimental setup. In Section V, we detail the performed experiments, present the obtained results and analyze the findings of the evaluation. Finally, Section VI closes the paper with final remarks and perspectives for future work.

II. RELATED WORK

Previous studies investigating the performance of flow table management can be classified in two categories: TCAM-only and multi-architecture based measurements. We describe each category as follows.

TCAM-only measurements. Chen and Benson [12] evaluate the impact of TCAM switches when dealing with control plane actions, considering real network traces and different types of SDN applications. However, the devices are evaluated through simulations using empiric models, which may not represent real switches precisely. Kuzniar et al. [9] do not provide an in-depth evaluation of flow table update latency. Specifically, the measurements do not consider the latency of rule insertion and deletion individually, which makes it difficult to understand how each operation impacts control plane performance. He et al. [8] present an extensive evaluation of switch performance. However, the employed methodology is unable to isolate the measured control plane latencies from potential noises introduced by other components, such as the packet generator.

Multi-architecture based measurements. Huang et al. [13] measure the query completion time for a specific application in different switches. Their investigation does not explicitly consider flow table update latency for the architectures in the paper. Sieber et al [10] evaluate TCAM and RAM switch update latency for rule insertion, using only simplistic scenarios (such as insertion of a single rule in an empty table). Lazaris et al. [11] perform measurements for two architectures (TCAM and RAM switches), but do not consider the operation of rule deletion by itself and provide a limited evaluation of a subset of devices.

Unlike previous work, in this paper we report results based on an extensive evaluation of different switch architectures. More specifically, we (i) are the first to identify and measure the influence of OpenFlow parameters in flow table update latency; (ii) define a robust methodology, which considers a range of evaluation scenarios and factors; and (iii) evaluate a broader set of switch architectures (TCAM-only, RAM-only and hybrid TCAM and RAM). We detail the evaluated architectures and the methodology in the next two sections.

III. SWITCH ARCHITECTURES

Nowadays, there is a vast diversity of SDN-enabled forwarding devices. The architectures of these devices differ with respect to implementation designs (software vs. hardware), use of memories (TCAM vs. RAM) and match-action structures (hash table vs. binary tree). In this section, we detail the existing architectures and then, the devices used in experiments.

A. Existing Architectures

We consider three SDN switch architectures, which represent a comprehensive set of devices: hardware switch using TCAM, software switch using RAM and hybrid devices.

TCAM hardware switch. This memory executes parallel lookup of the entire flow table at line-rate speed, providing fast packet classification. The efficient search, however, demands a high energy consumption, since all memory cells are activated at the same time for a single search in the table. In addition to the power consumption, the high cost per bit hinders large flow tables in TCAM switches [14]. In this context, network policies may need to be aggregated in wildcard rules. Note that, for some applications, this aggregation may potentially complicate (or, in some occasions, prevent) fine-grained management of flows.

RAM software switch. These devices are executed, usually, in general-purpose processors with a high amount of memory available. On one hand, the quantity of memory allows a fine-grained specification of network policies and flow management. On the other, the lack of dedicated hardware for packet forwarding leads to a lower transmission rate and higher CPU usage [15]. The limited throughput and inefficient CPU usage prevent the use of these devices in several contexts.

Hybrid hardware switch. TCAM switches are more efficient than RAM-based devices in terms of lookup response because of the ternary memory, but more expensive and typically limited in number of entries. Hybrid switches combine the use of TCAM and RAM tables and allow the utilization of reconfigurable match tables. For example, in a scenario which requires fine-grained control of flows, the number of rules to

Model	CPU	RAM	Table size (entries)
TCAM	CN5010 500MHz	512MB	2,5K
HTCAM	P1010 533MHz	2GB	2K TCAM
HRAM			32K RAM
RAM (Open vSwitch)	i3 3.10GHz	4GB	1-2M

TABLE I: Device specification

manage traffic can exceed TCAM capacity. In this case, it is possible to store rules in the RAM table. This diversity within the same device provides efficient resource utilization, albeit it is more difficult to predict performance since hybrid devices inherit the benefits and drawbacks of both TCAM and RAM.

B. Evaluated Devices

Below we define the set of switches considered, and then explain the differences and specific aspects of each design.

We measured the control plane latency of different switch designs when updating its flow tables: (i) TCAM (hardware switch with TCAM memory); (ii) RAM (software switch with RAM memory); (iii) HTCAM (hybrid hardware switch using only TCAM); and (iv) HRAM (hybrid hardware switch using only RAM exact-match tables). All switches were evaluated using OpenFlow 1.3. Due to a limitation of the hybrid device, it was not possible to evaluate the operational mode combining TCAM and RAM tables simultaneously. Table I presents information about the devices. Note that we do not claim that our results are exact for each switch design; our contribution lies in showing significant performance variability among architectures.

One of the designs we evaluate is a pure TCAM switch (i.e., we measure memory performance). The TCAM device has a single core processor, and an individual hardware table implemented using TCAM memory. This configuration forwards all packets which match with entries installed in the table at maximum speed. In order to evaluate the impact of the TCAM table, we disable all virtual tables in this switch during the experiments, that is, the switch does not store any tables in RAM. These software tables are implemented by vendors in some devices to increase the overall available table size. However, its poor performance directly affects flow configuration times.

In order to evaluate the RAM device, we use an instance of the Open vSwitch 2.5.4 LTS on a multi-core processor host, as it is the most accepted alternative. It is designed to be general purpose, working in commodity hardware. To achieve high performance using generic components, OVS makes use of extensive rule caching, keeping tables both in kernel and user space of the operating system [16].

The OpenFlow tables are stored in user space and serve to populate the cache tables placed in kernel space. The kernel tables are responsible for forwarding packets at high speed, avoiding the need to process all flows in user space. The significant disadvantage of this strategy is the requirement of cache revalidation whenever an OpenFlow table is modified. If cache verification is not performed, changes in user space tables are not reflected in kernel tables and lead to an inconsistent state of network policies. Depending on the rate of updates, the process of scanning and altering the cache may be complicated and may directly affect flow configuration time.

The HTCAM/HRAM device also has single core processor and allows the use of reconfigurable match tables offering different modes. Each profile holds a different table size according to the generality of entries, including tables for wildcard rules in TCAM, exact-match rules on hash tables in RAM or even modes designed for specific applications scenarios, such as NAT and IPv6 traffic. However, it is not possible to combine two or more modes simultaneously. This limitation means that we cannot take advantage of multiple profiles concurrently. In our evaluation, we analyze two modes of reconfigurable tables: single table using TCAM and a single exact-match hash table using RAM. These alternatives were chosen because they represent the extreme sides (more generic modes) of the hybrid switch. Moreover, by choosing these two modes, it is possible to compare the performance of two distinct memories and match-action structures, evaluating the benefits and drawbacks of each implementation.

IV. METHODOLOGY

We describe below the methodology used in the experiments. In order to accurately measure flow table update latency of different switches, we require a precise and synchronized environment (Section IV-A). We also need a strict setup for our experiments, considering the set of flows, workload and metrics (Section IV-B).

A. Evaluation Environment

The environment has four components: a network *controller*; a packet *generator*, which transmits traffic to the switch; the *switch* under evaluation; and a *receiver*, which receives the packets forwarded by the switch.

The topology, shown in Figure 1, is simple enough, enabling accurate experiments. Since we use information from both the controller and the receiver in the latency measurement, we run these two components along with the traffic generator on the same host to avoid the problem of fine-grained synchronization of clocks. We monitor the load on the host to ensure that there were neither resource containment nor interference between the three modules. The host is physically connected to the switch using three 1 Gbps interfaces. The network controller connects to the switch control port via interface `eth0`, sending control plane requests (*flow_mod*) to insert/modify/delete entries in the switch flow table.

The traffic generator uses `eth1` interface to send packets and the receiver uses `eth2` to receive them. Both interfaces (`eth1` and `eth2`) are connected to data ports in the switch. In the evaluation of the RAM device, the physical switch was replaced by a host with OVS. This host runs only an Open vSwitch instance and has three Gigabit interfaces to emulate switch data and control ports. We highlight that OVS is not running in a virtualized or emulated scenario, but on top of Linux with its kernel module enabled.

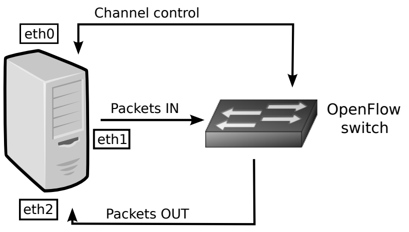


Fig. 1: Topology.

B. General Experimental Setup

Next, we describe the setup common to all experiments, such as rule match type, workload and metrics. We postpone the specifics of each experiment configuration until Section V, along with results, to improve readability.

Match type. Given the characteristics and limitations of each device, we considered two distinct sets of flow entries. The first is identified by having rules with *exact-match*. That is, all match fields must be filled with some value (no wildcards allowed): IP src/dst, src/dst port number, ToS, switch input port, EtherType and transport protocol. Since rules are exact, no more than one entry can match with a rule, and priorities are not needed. Usually, exact-match rules are stored in RAM (e.g., using hash tables and prefix trees) given that RAM memory is cheaper and allows for larger tables [17].

The second set contains rules with most match fields *wildcarded*. We use rules with exact values only in the IP src/dst and EtherType fields, as it is not possible to install rules with all fields wildcarded or with equal matches. For example, installing several rules with in_port specified with the same value and all other header fields wildcarded would implicate in the installation of a single rule in the flow table by the OpenFlow, as all rules would be equal. In this case, the use of priorities is strictly necessary because different entries may match with the same rule. This scenario is used to analyze three switch architectures: TCAM, HTCAM and RAM. We could not evaluate the HRAM design using this set of rules because its flow table is implemented as a hash table in RAM, only allowing the installation of rules with exact-match fields. Moreover, the table has support for a single priority, which makes it impossible to evaluate the scenarios with varying priorities.

Workload. It is divided into two parts: control plane and data plane. The former is responsible for modifying the flow table, and the latter, to detect the moment when the changes are actually performed in the device. The control plane load consists in a burst of *flow_mod* requests issued by the controller to install/modify/delete rules on the table. We employ bursts with sizes varying between 50 and 1950 requests, with the smallest table among all devices being at least as large as 2000 entries (Table I). By using this burst length, we guarantee that no switch flow table gets overflown.

The data plane workload, in turn, consists of a single flow starting from interface `eth1` of the host, passing through the switch and returning to the host via interface `eth2`. This flow is activated before the dispatch of the evaluated *flow_mod* burst and remains active until all table updates are completed. The packets of the flow match the most recent installed/modified rule. This entry does not necessarily have the higher priority among the other rules in the table. By using this workload, the number of control messages issued by the controller is reduced, which prevents overloading both the data and control planes of the switch.

The traffic generator sends UDP messages with constant inter-arrival time ($\simeq$ 10 μs) on `eth1` at a rate of 50 Mbps. The low throughput is intentional, since we aim at isolating the time taken by the switch to process updates in the flow tables (we do not evaluate the capacity/performance of the switch data plane, i.e., its forwarding rate). Like previous work, we confirm that the flow table was updated via data plane: a packet analyzer examines interface `eth2` of the host. This strategy is necessary to avoid inaccurate results [18], [9]. The packet analyzer has a microsecond precision; we confirm, via additional measurements, that it does not introduce any overhead to the measurements.

Metric. The metric we focus on is flow table update latency, similarly to [11], [10]. However, those papers include in the metric definition the transmission and propagation delays. We believe the update time should be based only on the time to modify the table in the device itself. For this purpose, we first measured the RTT between the network controller and switch ($\simeq$ 0.2 ms) and, assuming it is nearly constant, we subtracted the value (RTT) proportional to the number of *flow_mod*s emitted by the controller from the final results (e.g., $\simeq$ 400 ms when sending 1950 requests).

The plots in the next section show the combined time to execute a burst of rule insertion/modification/deletion operations. The x-axis always represents the burst size, that is, the number of operations being processed, while the y-axis represents the time necessary to perform all operations. We execute at least five redundant repetitions for each burst size, varying the x-axis in steps of 50 and show all points instead of the median. More specifically, when latency variability among repetitions was negligible, we executed only five repetitions (as more repetitions would yield the same results). When results showed high variability or unusual behaviors, we performed a higher number of repetitions and highlighted the details in the description. The source code of our experiments is available online [2].

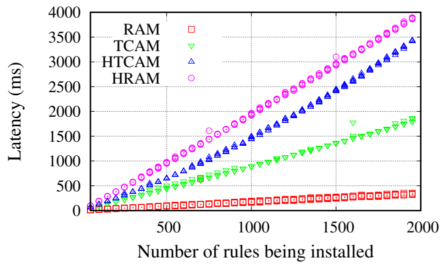
V. Evaluation

In this section, we detail the experiments and analyze the results. They are organized according to the three table update operations evaluated: rule insertion (Section V-A), modification (Section V-B) and deletion (Section V-C).

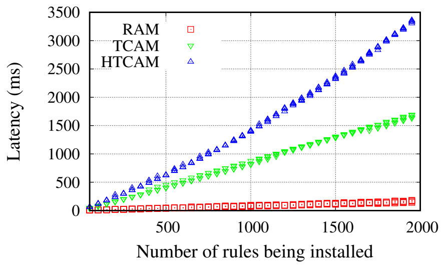
A. Insertion

To evaluate insertion, we investigate two factors that intuitively influence the time needed to install rules in the

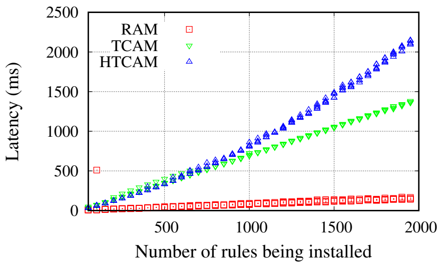
[2]https://github.com/fmmazz/SDN-table-update



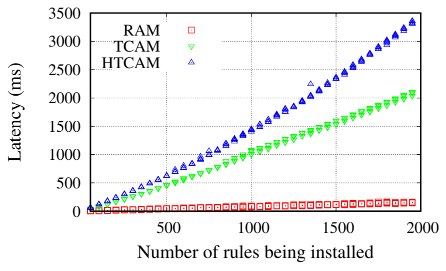
(a) Exact match entries



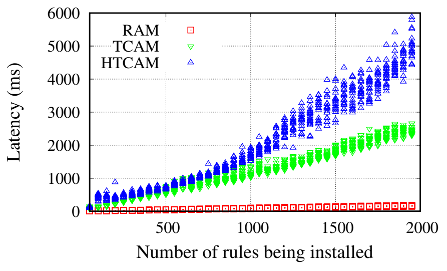
(b) Wildcard Equal priority



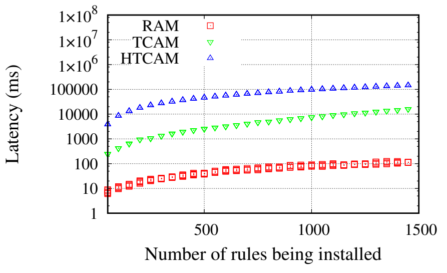
(c) Wildcard Incr. priority



(d) Wildcard Decr. priority



(e) Wildcard Random priority



(f) Priority intercalation

Fig. 2: Rule insertion.

flow tables: match type (exact or wildcarded) and priority choices. We understand that match type affects processing cost and, consequently, switch performance when inserting rules. The conclusions are obtained by comparing the plots of exact match and variations of wildcard match. The priority of the rule being installed (relative to other entries in the table), in turn, may result in the displacement of previously installed rules, which directly influences device performance when setting up new flows.

We describe the methodology used for rule insertion as follows. First, the device flow table is emptied. Then, we install a low priority rule that works as a table miss entry. That is, this rule matches any packet and has the action to drop all matching traffic; its role is to discard packets that do not match any entry in the table. Then, we install a burst of R rules with sequential IP dst. The action of these rules is to forward matching packets to the host. The insertion latency is defined as T_d - T_c, where T_c is the emission of the first *flow_mod* request of the installation burst and T_d the first packet observed via data plane in the host interface.

We use the set of rules with exact-match to understand the influence of the match type in rule insertion for all four architectures. All rules have the same priority. Figure 2a presents the results. In summary, we verify that (i) the RAM switch shows the best performance among all devices for rule insertion; (ii) there is a significant disparity of performance between devices with similar match-action structures (for example, HRAM and RAM switches); (iii) there is a notable contrast in rule insertion latency within the hybrid switch, depending on the mode (HRAM or HTCAM); and (iv) flow table update latency grows linearly according to the number of rules being installed. Next, we detail each finding.

The lowest latency for rule insertion was observed in the RAM switch. We explain these results by the extensive use of rule caching and the high number of optimizations implemented in the latest versions of OVS [16]. Even though not all rules are matched and installed in the fast path, the software switch still has to process and update all requests in the slow path. The use of multiple threads combined with batch processing of control plane requests that arrive together take advantage of multi-core processors to balance the load and to decrease the number of accesses to the memory. We verified through additional experiments that the RAM switch keeps the optimal performance even when installing a burst of 50K rules. Note that, because most traditional switches have specific processors with a single core, some OVS optimizations may not be used. Moreover, we explain the significant difference in insertion latency for switches with similar characteristics (HRAM and RAM) because of the difference in computational power.

The second finding to highlight in Figure 2a is the difference in rule insertion latency for the hybrid switch (comparing HRAM and HTCAM). Intuitively, HRAM (which has match-action structure and memory similar to RAM) would have a better performance when comparing to HTCAM. Surprisingly, we observe that HRAM has higher cost than HTCAM for all burst sizes (more than 500 ms for a burst size equal or greater than 750). When investigating the causes, we discovered the following: throughout the installation of the rule burst, the HRAM device processed only a few *flow_mods* for a short period of time, queuing subsequent ones. We understand that the reason for this behavior is the non-optimized implementation of hash tables, which needs to be continuously resized, introducing an additional overhead to flow table update.

Rule priority. In the second set of experiments, we seek to evaluate the impact of rule priority in the insertion latency.

We used the set of rules with some wildcarded fields and four priority patterns: equal, increasing, decreasing and random. The first one installs a burst of rules with equal priority, as indicated by its name. The increasing/decreasing patterns employ a burst of rules with unique priorities, following the order of their respective pattern. The last one (random) installs a burst of rules with non-exclusive priorities, varying from 1k to 15k. This interval was chosen to (i) reduce the chance that several rules end up with the same priority; and (ii) allow the representation of a more realistic scenario, in which the installed rules do not follow a predetermined order (such as increasing or decreasing).

Figures 2b, 2c, 2d and 2e present the results for the different priority patterns. Recall that the evaluation in this case is restricted to the three architectures that allow the use of varying priorities. In summary, we observe that (i) rule insertion with random priority (Figure 2e) shows higher latencies than other patterns (note that the y-axis in the plot is wider) and higher variability; (ii) the decreasing pattern (Figure 2d) exhibited higher latency than increasing priority (Figure 2c); and (iii) the insertion latency grows linearly according to the number of rules being installed for all devices, regardless of the priority pattern. Next, we detail each of these findings.

Rule insertion with random priority (Figure 2e) presented higher latencies when compared to other patterns, in contrast the results found by Lazaris et al. [11]. For this experiment, we performed 20 redundant rounds for each device. We believe the results obtained in that study are not correct due to the simplistic methodology adopted by the authors. We also observe high variability in the results for the HTCAM and TCAM devices. When investigating[3], we detected that the increased latency and variability are associated to the randomness in the priority of the rules being installed (between 1k and 15k). This variation creates a significant overhead to the process of search, reordering and installation of rules in the flow table: reordering can lead to the displacement of a considerable number of entries for each new rule installation.

The insertion with increasing priorities (Figure 2c) results in lower latencies when compared to the decreasing pattern (Figure 2d), in line with a similar experiment in Lazaris et al. [11]. When comparing the plots, we verify that the performance of HTCAM and TCAM falls considerably when installing rules with decreasing priority. Latency for the decreasing pattern increases substantially compared to the increasing priority: for a burst of 1950 requests, around 1500 ms for HTCAM and 700 ms for TCAM. The results from the equal priority scenario (Figure 2b) are similar to the decreasing pattern (Figure 2d). We note that insertions are slower because rules may need to be shifted on the table. We conclude that the memory employed in these devices may not be prepared to install rules with a decreasing priority pattern. In contrast, we notice that latency for the RAM device is not affected. This happens because the switch implements its flow tables as hash tables (which do not order entries by priority), so their rules do not need to be shifted and, consequently, their performance is not influenced.

[3]We discussed this fact with the switch vendor engineers, but we cannot make the discussions publicly available due to the signature of a non-disclosure agreement.

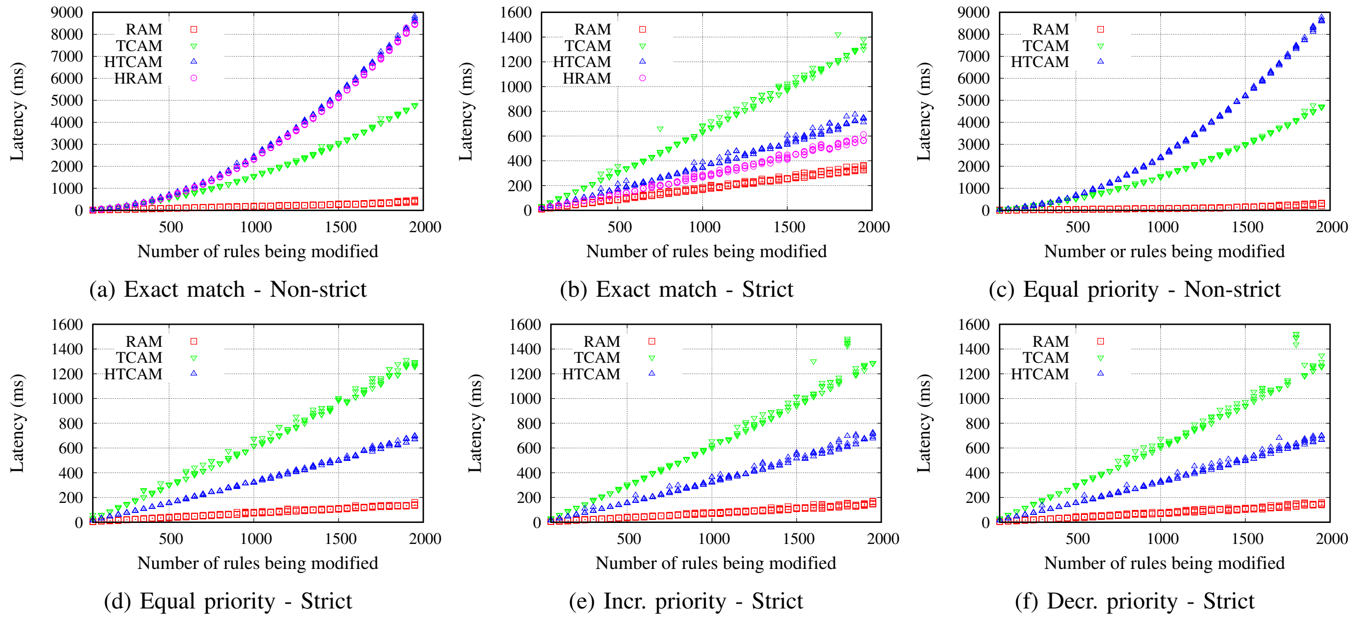
Priority intercalation. To better understand the effect of priorities and table occupation in rule insertion latency, we performed a third set of experiments. In this case, the rules to be inserted have different priorities in comparison to the rules already installed in the flow tables (again, recall that the use of priorities limits the comparison among three architectures).

In this set, we performed two experiments. Next, we describe the methodology used in the first one. Initially, the switch flow table is emptied, and 500 rules are installed with a 'low' priority of 1,000. Then, the switch receives a burst of R installation requests, containing rules with 'high' priority (32,000). We evaluated the time to install all rules considering bursts of 50 to 1450 requests. These burst lengths were chosen considering the size of the smallest switch table (Table I). The action associated with these rules is to forward traffic to the host. The second experiment is the same but with inverted priorities: rules with low priority are installed in a table already populated with high priority entries.

Figure 2f presents the results for the first experiment, using logarithmic scale for the y-axis and linear scale for the x-axis. We observe that the difference in flow table update latency between devices is at least one order of magnitude and can reach up to three orders with a burst of 1,000 requests (HTCAM and RAM). The latency for the HTCAM exceeds 10 seconds even with a short burst (150 insertions). The explanation for the high latencies is the excessive overhead to process, reorder and install the updates in flow tables at line rate. We perceive that memory management in these devices is unable to deal appropriately with priority diversity and, consequently, these devices are incapable of installing rules efficiently.

The results of the second experiment with priority intercalation are very similar to rule insertion with increasing priority (Figure 2c) and therefore, omitted. Table update latency grows linearly according to the number of rules being inserted. The values for TCAM and HTCAM show that the devices tend to install low priority rules in the last address of memory, in line with a similar experiments in He et al. [8]. This ordering is used to avoid rule displacement when inserting low priority rules in a table populated with high priority entries. Once again, it is possible to observe that the RAM device is not influenced by different priorities.

Implications. The results show how limited knowledge of the switch operation (considering different architectures) can impact overall network performance. Understanding how switches perform with different configurations (e.g., rule priority and match type) is essential to avoid the use of an ineffective and, perhaps, incorrect network control plane. In real SDN networks (where a controller manages several switches), installing a burst of rules in all these devices with a poorly configured control plane may considerably increase flow setup time, negatively impacting the performance of time-sensitive applications and the state of the entire network. In general, our results help identifying potential inefficiencies, which should



(a) Exact match - Non-strict (b) Exact match - Strict (c) Equal priority - Non-strict

(d) Equal priority - Strict (e) Incr. priority - Strict (f) Decr. priority - Strict

Fig. 3: Rule modification.

be avoided to ensure efficient network performance.

B. *Modification*

Similarly to the insertion, we first use the set of exact-match rules and fixed priority to evaluate the modification for the four designs. Then, we evaluate three architectures considering the impact of different rule priorities.

Unlike previous studies, we consider the impact of the OpenFlow parameter *strict*. As the name implies, it forces the match of entries to be 'strict' or exact, considering the priority and all fields of the *flow_mod* match. Thus, when *strict* is used, at most one entry is modified by a request (if not used, multiple entries may be modified with a single request).

Next, we describe the methodology used in the modification experiments. First, the switch flow table is emptied and, then, R initial rules are installed. The action associated with these rules is to forward traffic to an unused port of the switch, which will drop all matching packets. Later, a burst of R modification requests is sent to the switch, altering the initial rules actions to forward traffic to the switch output port connected to the host. The modification latency is defined as T_d - T_c, where T_c is the emission of the first *flow_mod* request of the modification burst and T_d is the first packet observed via data plane in the host interface.

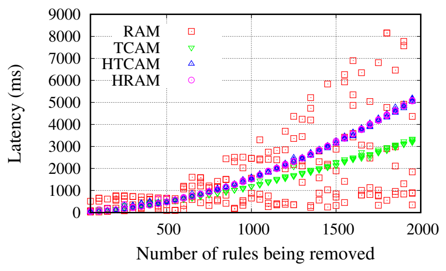
The first experiment is based on exact-match and fixed priority. Figures 3a and 3b present the results for the four architectures (with and without the parameter *strict*). In summary, we observe that (i) the use of the *strict* provides a significant decrease in the table update latency (note that the y-axis is very different), reaching 15x for the HRAM device and up to 11x for the HTCAM switch; (ii) when not using *strict*, latency grows exponentially according to the number of rules being modified for the TCAM, HTCAM and HRAM devices, but linearly for the RAM one. Next, we detail the results.

When comparing the plots of Figures 3a and 3b, it is evident the impact of the *strict* parameter (note the y-axis). The reason for this difference is explained as follows. Without the *strict*, a modification request demands a complete search in the entire table, because multiple entries may match with the information contained in the *flow_mod*. Even for hash table match-action structures (HRAM and RAM), there is a complete lookup on the table. The huge disparity of latency between these two devices is due to the difference of computational power, in line with what was discussed previously. The use of a single core processor directly influences the cost of a complete table search. In contrast, with *strict*, the operation is terminated as soon as a (strict) match occurs, which considerably decreases table update latency.

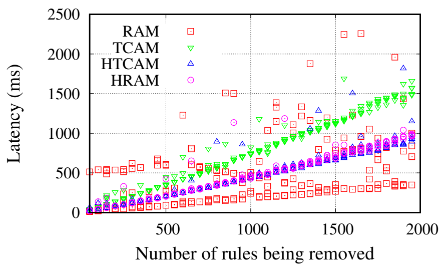
Rule priority. We used a set of experiments to evaluate the performance of modification operation when rules have different priorities (and wildcard masks). We considered three different priority patterns: equal, increasing and decreasing. Within an experiment, we use the same pattern and rule priority both to install the initial rules in the table and for the burst of modification requests issued by the controller, as our goal is to modify the same rules that are already installed in the device.

Figures 3c, 3d, 3e, 3f show the results. In summary, we observe that (i) the *strict* parameter has the largest impact when modifying rules with distinct priorities; (ii) the priority pattern does not affect the time to process modification requests; and (iii) rule match type does not influence rule modification latency.

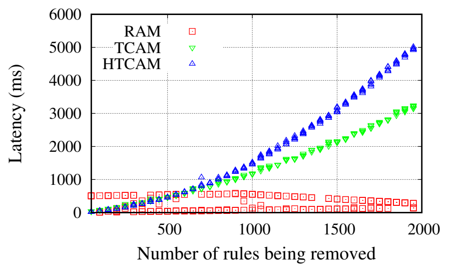
The RAM switch showed a linear growth according to the number of rules being modified, independently of the priority



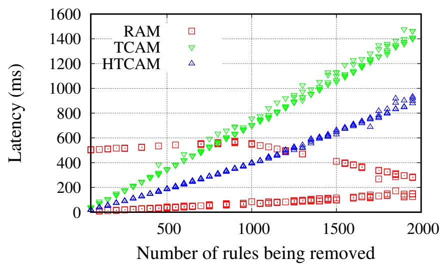
(a) Exact Match - Non-strict



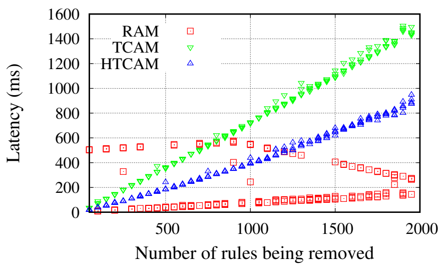
(b) Exact Match - Strict



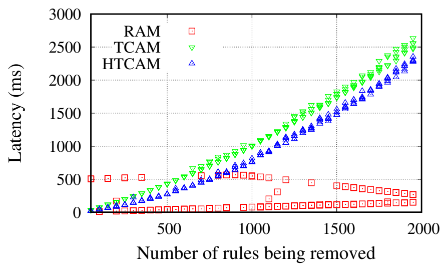
(c) Equal priority - Non-strict



(d) Equal priority - Strict



(e) Incr. priority - Strict



(f) Decr. priority - Strict

Fig. 4: Rule deletion.

pattern and *strict* usage. In HTCAM and TCAM devices, the absence of the parameter implies an exponential growth of the latency, according to the number of rules being modified. In contrast, with the parameter, latency increases linearly according to the number of rules being modified.

We notice that rule modification is not affected by the priority pattern or match type. The experiments show that, for any variation of these factors, the results remain similar and present identical behavior. He et al. [8] indicate that rule modification is not affected by rule priority. However, the results of that study fail to consider the *strict* parameter. When it is off, rule priority is never examined and, evidently, the results will be the equivalent for any priority pattern used.

Implications. The results show the importance of an accurate configuration of the network control plane. The right understanding of the switch architectures, combined with the use of OpenFlow parameters, can substantially impact overall network performance. Improper usage of parameters may significantly influence flow setup time and hinder the application of network policies. Our results improve the understanding of control plane configuration and benefit network operators.

C. Deletion

In line with insertion and modification, we first use exact-match and fixed priority rules to measure the deletion operation for all four architectures. Next, we evaluate deletion in three architectures (RAM, TCAM and HTCAM) considering the impact of different rule priorities with and without the OpenFlow *strict* parameter. Unlike previous work, we evaluate the influence of the *strict* parameter and match type in rule deletion latency, as well as analyze the performance of entry deletion in a RAM switch.

We describe the methodology used for the first set of experiments (with deletion using exact-match and fixed priority) as follows. First, the switch flow table is emptied, and R initial rules are inserted. The action associated with these rules is to forward traffic to the host. Then, a burst of R deletion requests is sent to the switch. The deletion latency is defined as T_d - T_c, where T_c is the emission of the first *flow_mod* of the deletion burst and T_d the last packet observed via data plane in the host interface. This methodology was adopted considering the rule priority limitation of the HRAM device, which prevents the installation of a low priority rule responsible for processing packets that do not match any entry in the table.

Figures 4a and 4b show the results. In summary, we observe that (i) the *strict* parameter is the aspect with the greatest impact on rule deletion delay, causing a decrease in latency of 2x for TCAM and up to 5x for HRAM and HTCAM (Figure 4b); and (ii) latency increases linearly according to the number of rules being removed in the TCAM, HTCAM and HRAM devices.

Surprisingly, there was considerable variation in the results of the RAM device. We investigated the issue and discussed the possible causes in the OVS mailing list. We conclude that the behavior is due to the architecture of OVS: a substantial number of threads is created to accomplish the task of removing a large number of rules, increasing the degree of concurrency among the threads responsible for obtaining statistics and cache revalidation. This behavior was observed even with 30 redundant repetitions.

Rule priority. In the second set of experiments, we used the wildcard match rules and three distinct priority patterns: equal, increasing and decreasing. We describe the methodology applied to rule deletion with varying priorities in the following manner. First, the switch flow table is emptied. Next, a low priority rule is installed, which matches with any packet and has the action to forward traffic to the host. Then, R initial rules are installed using one of the priority patterns, with the associated action to drop matching traffic. The packet generator is activated, and the traffic matches with the last rule installed (which discards these packets). Later, a burst of R deletion requests is issued by the controller using the same pattern and rule priority of the initial rules, as we would like to remove the already installed rules. It is possible to observe the moment that the rules are removed from the table because packets stop matching with the last installed rule and start matching with the low priority rule. The deletion latency is defined as T_d - T_c, where T_c is the emission of the first *flow_mod* request of the deletion burst and T_d is the first packet observed via data plane in the host interface.

Figures 4c, 4d, 4e and 4f present the results. Once again, it is clear the importance of the *strict* parameter when comparing the plot of Figures 4c and 4d (note the differences in the y-axis scale). Regardless of the priority pattern used to remove rules and match type, the behavior for TCAM and HTCAM devices was quite similar, increasing the latency linearly according to the number of rules being removed. We also noticed that, for these devices, exact-match and wildcard masks do not affect the table update latency. Besides, we found that removing rules with decreasing priority in these switches takes longer when compared to the other two patterns (equal and increasing), even with *strict*. We conclude that this behavior is due to the displacement of rules in TCAM tables. As previously mentioned, we verified that the TCAM memory of the devices may not be prepared to search for rules with a decreasing priority pattern.

Regarding the RAM device, we observe a very unusual bimodal behavior in all priority patterns despite the use of *strict* (note what appears to be two curves for the RAM device in Figures 4c, 4d, 4e and 4f). This behavior was observed even with 60 redundant repetitions. We contacted the OVS developers and discussed these results. The conclusion is that the variability is due to the process of cache revalidation performed by the switch. Whenever the OpenFlow tables are modified, OVS needs to iterate over all entries in cache tables (kernel space) to check if they need to update their actions or they remain correct. To perform an optimized execution, OVS batches successive *flow_mod* requests to execute several operations at once. Besides, it delays cache revalidation briefly when two table changes occur in rapid succession, thus generating the bimodal behavior. In contrast, rule deletion with wildcard masks did not show the high variability presented in exact-match (comparing Figures 4a and 4b with Figures 4c, 4d, 4e and 4f). This can be explained by the substantial load reduction of the switch when it discards all packets, instead of forwarding them through an output port.

Implications. These results show unusual behaviors previously unknown in the literature for RAM switches. Again we show the relevance of understanding switch operation (how each architecture behaves when dealing with control plane requests) and its parameters. Our findings show scenarios that could be critical to networks that depend on RAM switches, as rule deletion presents high latencies with significant variability. In general, these behaviors may directly affect overall application and network performance. Also, our findings may contribute to improvements in OVS to be made by the community.

VI. FINAL REMARKS

In this paper, we performed a comprehensive measurement of switch table update latency, comparing four SDN-enabled forwarding device designs. The main contributions of this paper are as follows. First, we evaluate a broader set of switch architectures (TCAM-only, RAM-only and hybrid TCAM and RAM) than related work and we are also the first work to identify and measure the influence of OpenFlow parameters in flow table update latency. Second, we define a robust methodology, which considers a range of evaluation scenarios and factors. Third, results indicate that (i) latency disparity between devices can scale up to three orders of magnitude for rule insertion; (ii) OpenFlow parameters inflate flow setup time up to 10x and 6x for rule modification and deletion, respectively; and (iii) OVS has unusual behaviors (and high variability) for rule deletion. We believe that our findings highlight the importance of understanding the operation of switches with different architectures and the need for an accurate configuration of the control plane to obtain the best performance of SDN networks.

As future work, we intend to (i) explore new switch architectures (e.g., with programmable data planes); (ii) investigate switch performance using real network workloads; and (iii) measure the performance of both control plane operations (*flow_mod*) and data plane requests (table lookups).

ACKNOWLEDGEMENTS

This work has been supported by Project Phoenix (MCTI/CNPq/Universal 460322/2014-1) and Project P4Sec (RNP-NSF Cybersecurity).

REFERENCES

[1] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] A. Roy *et al.*, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Aug. 2015.

[3] T. Benson *et al.*, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280.

[4] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74.

[5] K. Jang *et al.*, "Silo: Predictable message latency in the cloud," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 435–448.

[6] T. Koponen *et al.*, "Network virtualization in multi-tenant datacenters," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 203–216.

[7] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[8] K. He *et al.*, “Measuring control plane latency in sdn-enabled switches,” in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR ’15, Santa Clara, California, USA, June 17-18, 2015*, 2015, pp. 25:1–25:6.

[9] M. Kuźniar *et al.*, *What You Need to Know About SDN Flow Tables*. Cham: Springer International Publishing, 2015, pp. 347–359.

[10] C. Sieber *et al.*, “How fast can you reconfigure your partially deployed SDN network?” in *Networking*. IEEE, 2017, pp. 1–9.

[11] A. Lazaris *et al.*, “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*, ser. CoNEXT ’14. New York, NY, USA: ACM, 2014, pp. 199–212.

[12] H. Chen and T. Benson, “The case for making tight control plane latency guarantees in sdn switches,” in *Symposium on SDN Research*, ser. SOSR ’17. New York, NY, USA: ACM, 2017, pp. 150–156.

[13] D. Y. Huang *et al.*, “High-fidelity switch models for software-defined network emulation,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’13. New York, NY, USA: ACM, 2013, pp. 43–48.

[14] G. J. Narlikar *et al.*, “Coolcams: Power-efficient tcams for forwarding engines,” in *Proceedings IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Francisco, CA, USA, March 30 - April 3, 2003*, 2003, pp. 42–52.

[15] M. Honda *et al.*, “mswitch: A highly-scalable, modular software switch,” in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR ’15. New York, NY, USA: ACM, 2015, pp. 1:1–1:13.

[16] B. Pfaff *et al.*, “The design and implementation of open vswitch,” in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 117–130.

[17] C. Yu *et al.*, “Characterizing rule compression mechanisms in software-defined networks,” in *International Conference on Passive and Active Network Measurement*. Springer, 2016, pp. 302–315.

[18] C. Rotsos *et al.*, “Oflops: An open framework for openflow switch evaluation,” in *Proceedings of the 13th International Conference on Passive and Active Measurement*, ser. PAM’12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 85–95.