Performance Study of P4 Programmable Devices: Flow Scalability and Rule Update Responsiveness

Hasanin Harkous†, Mu He†, Michael Jarschel†, Rastin Pries†, Ehab Mansour*, Wolfgang Kellerer*

**Technical University of Munich*, firstname.lastname@tum.de
†*Nokia Bell Labs*, firstname.lastname@nokia.com
Munich, Germany

***Abstract*—Networking devices with programmable data planes, such as P4 programmable devices, are gaining more popularity because of the flexibility they provide in describing the packet processing behavior. Despite this attained flexibility, the performance of these devices can be the Achilles' heel in case the desired performance level is not met.**

To this end, we evaluate the performance of three state-of-the-art P4 devices focusing on the following properties: (i) the device's processing latency as a function of a scaled number of flows; (ii) the device's response time in reaction to control plane commands. The scalability analysis shows that different devices have different limits on the maximum number of flows they can support. On the other hand, the device's response time to control plane commands is found to be in milliseconds, which is three orders of magnitude larger when compared to the measured data plane's packet processing latency.

***Index Terms*—Performance Evaluation, Programmable Data Planes, Flow Scalability, Rule Update Responsiveness**

I. Introduction

Emerging applications with stringent requirements pose a significant challenge to current networks. The advent of network programmability is considered a promising answer to this challenge by enabling the adaptation of network behavior based on connectivity requirements. Software Defined Networking (SDN) separates the control plane from the forwarding devices and enables control plane programmability. The control plane in turn pushes rules to the data plane to update the forwarding behavior. As a complement, a Programmable Data Plane (PDP) enables the customization of the forwarding device, specifically its packet processing pipeline. In [1], P4 is introduced as a domain-specific programming language and has become a promising candidate to realize the PDP concept. Any device that can execute P4 programs is called a P4 target. Due to P4's feature of target-independence, multiple types of P4 targets are introduced to enable packet processing with both software and dedicated hardware. For example, *(i)* T4P4S [2] is a framework that generates a C program that can be supported by several general-purpose CPU platforms with the acceleration capability offered by DPDK. *(ii)* NetFPGA-SUME [3] leverages the programmability of FPGAs to provide low-latency and high-throughput concurrent packet processing. *(iii)* Netronome SmartNIC belongs to the category of Network Processing Units (NPUs), which comes with tens of cores with high parallelism capability and is optimized specifically for packet processing tasks.

While P4 programmable devices can play an important role in addressing the issues accompanied by the ever-growing demand for more complex applications, it is equally important to understand the performance of these devices and their limitations to assess their applicability for different use case scenarios. For example, benchmarking and understanding the performance of P4 programmable devices can be a valuable input for designing P4-based time-sensitive networks with deterministic performance requirements. Moreover, identifying the performance limits of these devices helps in assessing the convenience of using these devices for different use case scenarios.

To this end, we evaluate the performance of the three aforementioned P4 programmable devices focusing on two important criteria that were not addressed in the literature, namely flow scalability and rule update responsiveness. The flow scalability analysis determines the maximum number of flows supported by a P4 device, and the performance variation when the number of flows scales up. As an example, if a P4 table implements access control functionality, then the maximum number of rules that can be installed in this table limits the number of defined users. On the other hand, the rule update responsiveness analysis reveals the time for a packet processing pipeline to adapt in response to control plane commands. Characterizing such delay is important as the state in the control plane is inconsistent with that at the data plane over this period, which may lead to inconsistent forwarding behavior.

The remainder of this paper is structured as follows. The description of the flow scalability and rule update responsiveness experiments are provided in Section II. Results of these experiments are shown in Section III. Section IV reviews related literature. Finally, Section V concludes the work.

II. Testbed & Experiment Design

In this section, we describe the experiments designed to evaluate the P4 devices' processing latency as a function of traffic flow scalability, as well as the response latency to control plane update rules.

A. Flow Scalability

The purpose of the flow scalability study is to identify the performance and limits of the investigated devices when the

ISBN 978-3-903176-39-3© 2021 IFIP

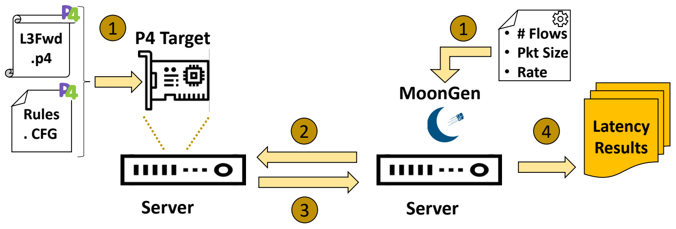


Fig. 1: Testbed setup adopted for flow scalability experiments. The P4 target is installed into a server. MoonGen generates and sends traffic to the P4 target, where packets are processed, and then analyzes the received packets' latency.

incoming traffic flows scale up. First, we describe the testbed built for this evaluation, then we elaborate on the conducted experiments.

1) Testbed Setup: Fig. 1 shows the adopted testbed. Three different P4 programmable packet processors, each belonging to a different class of platforms, are installed into a server: Agilio CX 2x10GbE Netronome SmartNIC and Xilinx Virtex-7 XC7V690TFFG1761-3 NetFPGA-SUME are plugged into the server's PCIe bus, while T4P4S-DPDK open-source software switch runs on the server. The hosting server in the case of Netronome SmartNIC and T4P4S software switch is Nokia NDCS16RM AirFrame Compute Nodes with 16 cores (dual-socket Intel Xeon CPU E5-2630 v3 @ 2.40GHz) and 64GB of 2133 MHz DDR4 memory, while it is another server with 6 Cores (Intel CPU i7-8700 @ 3.20 GHz) and 32GB of 2400 MHz DDR4 memory in the case of NetFPGA-SUME. Due to logistical reasons, we were unable to use the same servers for all experiments, but this does not impact the comparability of the results of the hardware P4 devices as these servers only host the P4 devices without impacting the packet processing taking place in the P4 device.

In each scenario, the layer 3 forwarding (L3Fwd) network functionality, written as a P4 program, is loaded into the P4 device under test along with the forwarding rules. Then, the MoonGen packet generator [4] is used to generate traffic with a configurable number of flows, packet sizes, and bit rates. The generated traffic traverses a 10 Gbps link to reach the P4 target where packet processing takes place. Then, the packets are returned on another 10 Gbps link to the traffic generator, where the per-packet latency is reported.

In every measurement case, MoonGen generates the traffic at the line rate of the tested P4 devices, i.e., the maximum rate which can be handled by a device before packets are dropped. The line rate (with framing) adopted in the case of Netronome SmartNIC and NetFPGA-SUME is 10 Gbps, and that for the T4P4S software switch is 9.7 Gbps. The size of packets is set to three different values: 256 Bytes, 1000 Bytes, and 1500 Bytes to study the impact of differences in packet size on the collected results. The latency results of each test are based on the collected per-packet latency of 100 thousand generated packets.

2) Experimental Scenarios: As works in [5] and [6] already characterize the impact of the complexity of the P4 program on packet processing latency, the conducted experiments in the flow scalability evaluation consider a single P4 program and focus on pushing the match-action units (i.e. tables) of the investigated P4 devices to their limits. Identifying these limits is crucial as they can determine, based on the anticipated traffic load, whether a device is suitable for a specific scenario or not. In this evaluation, we consider a Layer 3 Forwarding (L3Fwd) P4 program with a single table matching on the IPv4 destination address field and forwarding packets accordingly. The maximum number of rules that can be installed into this table determines the maximum number of routes that can be supported by this device. In the evaluation, we consider both exact and wildcard, i.e. Ternary or Longest Prefix Match (LPM), matching. First, we start with a basic case, where we only install a single rule to the routing table and generate a single flow with an IPv4 destination address that matches the installed rule. Then, we increase the number of rules to be installed into the table up to the maximum possible number of rules, while keeping the number of generated flows fixed to one. In the third and fourth cases, we keep the table filled with the maximum number of rules, while gradually increasing the number of generated flows to reach 1000 flows then finally the number of installed rules. Note that different flows are distinguished by unique destination IP addresses, and in all cases, we make sure that the generated flows match the installed forwarding rules. The four designed experiments with their IDs are summarized in Table I.

TABLE I: Experiment IDs of the considered cases in the flow scalability evaluation.

Experiment ID	**Num. of Rules**	**Num. of Flows**
R_1F_1	1	1
$R_{max}F_1$	Max	1
$R_{max}F_{1k}$	Max	1000
$R_{max}F_{max}$	Max	Max

In the P4 language, the size of a table is one of the parameters that can be specified when instantiating a P4 table. However, we observed that this "table size" parameter is different from the real maximum number of table entries that can populate a table. To confirm the maximum operable table size for each device and matching type, we configure the table size to the corresponding theoretical maximum and test the switch operation. If the switch does not operate successfully, we gradually decrease the table size and loaded rules in powers of two. Note that LPM is used as the wildcard matching type except for the case of NetFPGA-SUME where LPM is not supported. There, ternary matching is used instead.

In the case of Netronome SmartNIC, the table size in a P4 program can be set to values as large as 50 million entries without compilation errors, although the official documentation states that the maximum number of entries is 64k [7]. However, only 48k rules for both exact and wildcard matching could be installed on the device without causing loading errors. Note that we observe that this limit also depends on the size of the matching fields and action parameters in the targeted table. Tables have a limited memory capacity that can be saturated

TABLE II: Summary of the theoretical and operable maximum number of rules adopted in flow scalability evaluation for different P4 devices; exact (=), wildcard (*).

P4 Target	Match	Theo. Max. Rules	Oper. Max. Rules
Netro. SNIC	= and *	64k	48k
NetFPGA	=	512k	64k
	*	4k	4k
T4P4S	= and *	1k	1k

based on the product of the number of entries and the size of each entry.

NetFPGA-SUME uses the Xilinx P4-SDNet tool for compiling P4 programs. This compiler reports that the maximum table size is related to the match type [8]. For each match type, if the table size is configured with a value higher than the limit, it will be rejected by the Xilinx P4-SDNet compiler and result in a compilation error. However, a successful compilation does not guarantee a successful switch operation as there might be other issues that cause the switch to be inoperable. In the case of exact matching, a table size of 512k fails at the synthesis phase because of insufficient resources as the design requires more RAM. For table sizes of 128k and 256k, the control plane program could not add forwarding rules properly, although the synthesis process executes successfully. This happens likely because of the high utilization of Block RAM (more than 90%). Thus, we set the operable limit of the maximum number of rules, i.e., table size, to be 64k rules. In the case of wildcard matching, the documented limit of the table size for ternary matching, i.e., 4k rules, is identical to the operable one.

For the T4P4S software switch, the operable limit of the number of rules for both exact and wildcard matching types is equal to 1024 rules, although the configured table size can take greater values. This limit is hardcoded in the ”HASH_ENTRIES” and ”LPM_MAX_RULES” constants in ”dpdk_tables.h” [9]. Although installing more rules than this limit does not result in compilation or execution errors, we observe that only the last 1024 rules are actively adhered to in forwarding decisions. Note that the hardcoded limit could be changed, but in this evaluation, we stick to the default implementation of this switch.

The theoretical and operable limits of the number of rules for all the considered cases are summarized in Table II. In the flow scalability evaluation, we use the operable limits to set the maximum number of rules to be installed into the investigated P4 devices and the maximum number of flows to be generated by the traffic generator.

B. Rule Update Responsiveness

In this experiment, we focus on evaluating the time required for a control plane rule to take effect on the data plane, i.e., the P4 device’s response time to control plane rules. We will first describe the testbed to conduct this evaluation, and then explain the considered scenarios.

1) Testbed Setup: The servers described in the previous subsection are used to build the testbed to conduct this evaluation as shown in Fig. 2. Each P4 hardware target is plugged into the PCIe interface of a server, through which the control plane communicates with the P4 target. Netronome SmartNIC and NetFPGA-SUME have both physical and virtual interfaces. MoonGen is connected to one of the physical interfaces of the investigated P4 hardware targets, i.e., Netronome SmartNIC and NetFPGA-SUME. Note that this evaluation is not possible on the T4P4S software switch because its current implementation does not fully support adding control plane rules during runtime as the current implementation of this feature is still experimental. First, a P4 program with forwarding functionality and an initial rule that forwards incoming traffic on the physical interface (nf0_phy) to one virtual interface (nf0_vir) is loaded on the P4 target. Then, the packet generator is used to send traffic with configurable rate and packet size to the connected P4 target’s physical interface. The control plane running on the hosting server sends an update rule, over the control channel of the P4 target, to change the egress port of the outgoing traffic to virtual interface 1 (nf1_vir) instead of virtual interface 0. A timestamp before and after issuing the control plane update rule is taken and recorded as T1 and T2 respectively. Tcpdump is used to capture the received packets on the two virtual interfaces. The timestamp corresponding to the last packet received on nf0_vir before the rule update takes effect is recorded and denoted as T3. The timestamp corresponding to the first packet received on nf1_vir after the update rule takes effect is recorded as T4. Note that we make sure that all timestamps are taken by the same server to avoid time synchronization issues. Each test case is repeated 20 times, and T1, T2, T3, and T4 are collected to analyze the responsiveness of the data plane to control plane update rules. The control plane command to update the rule is issued with the default tools provided by the investigated P4 devices. The executable ”rtecli” tool issues the rules in the case of Netronome SmartNIC, while a provided python API is used in the case of NetFPGA-SUME. This API includes the relevant libraries to communicate the update command to the targeted device. Note that we add one additional timestamp into the provided tool’s source code for NetFPGA-SUME before the last line of code responsible for pushing the control plane command. This is done to quantify the tool’s pre-processing time.

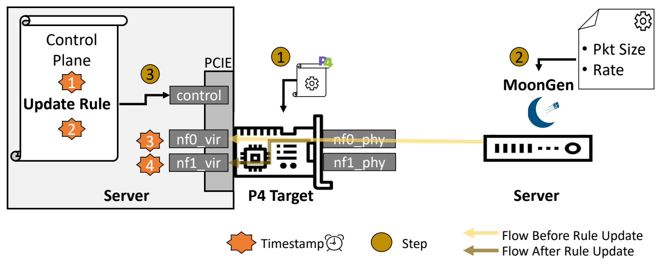


Fig. 2: Testbed setup adopted for rule update responsiveness experiments. The P4 device is installed into a server. MoonGen generates and sends traffic to the P4 device, where traffic is forwarded to the respective virtual interface based on the loaded control plane rule.

TABLE III: Cases and parameters considered in the rule update responsiveness experiments.

Variable	**Value**
P4 Targets	Netronome SmartNIC, NetFPGA-SUME
P4 Pipelines	Fwd_Exact, Fwd_Wildcard, Fwd_Register
Packet Size (in Bytes)	256, 1000, 1500
Rate (in Mbps)	100, 250, 500/ 6000
Metrics	Update_Rate, Response_Time

2) Experimental Scenarios: In the conducted experiments, we vary the traffic characteristics and the type of rules updated by the control plane while recording the following metrics:
(i) Update_Rate: This is the inverse of the time for a control plane command to execute and return, i.e., $(T2-T1)^{-1}$. This value determines the maximum rate at which update rules can be sent based on the minimum duration of consecutive control plane commands, i.e., $(T2-T1)$.
(ii) Response_Time: This is the time since the control plane command to update rules is issued until it takes effect on the packet processing behavior at the data plane, i.e., $T4-T1$. Over this period, the state at the control plane is inconsistent with that at the data plane, which may lead to outdated forwarding behavior.

We perform the measurements for three P4 pipelines to cover different types of update rule commands. These pipelines are Fwd_Exact, Fwd_Wildcard, and Fwd_Register. The first pipeline is made up of an exact match-action table, where exact rules are applied to change the egress port from nf0_vir to nf1_vir, while the second pipeline performs this operation based on a wildcard rule. The third pipeline updates the egress port based on the value stored in a register, where this value is updated by the control plane command.

The measurements were conducted while generating traffic with different characteristics. Three different packet sizes are considered: 256, 1000, and 1500 Bytes. Although the physical interfaces of the investigated devices support 10 Gbps traffic, the rate that can traverse the PCIe bus to reach the virtual interfaces is limited below that value. This maximum is only 500 Mbps in the case of NetFPGA-SUME due to the implementation of the "Reusable Integration Framework for FPGA Accelerators" [10], and approximately 6000 Mbps in the case of Netronome SmartNIC for small-sized packets. Accordingly, these two rates were selected as the maximum rates to be used in this evaluation, along with 100 Mbps and 250 Mbps as average- and low-load cases. Table III summarizes the details of this experiment.

III. Measurement Results

In this section, the measurement results of the flow scalability and rule update responsiveness experiments described in Subsections II-A and II-B are presented in Subsections III-A and III-B respectively.

A. Flow Scalability

Fig. 3 shows the box-plots of the measured packet latency on different P4 devices in μs for the different flow scalability experiments defined earlier when exact and wildcard (shaded) matching is used. The box-plots include the minimum, first quartile, median, third quartile, maximum, and outliers of the measured per-packet latency. The results show packet size equal to 1500 Bytes, i.e., the Maximum Transfer Unit (MTU).

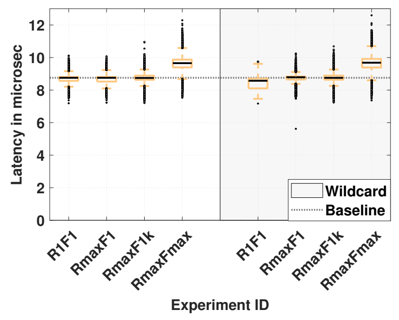
From Fig. 3a, we can observe that the median of the measured packet latency in the case of Netronome SmartNIC is equal to 8.7 μs when the number of rules and incoming flows is equal to 1 (i.e R_1F_1). Compared to this baseline case, the measured latency stays constant when the number of installed rules increases to its maximum value (i.e. $R_{max}F_1$). However, as the number of incoming flows increases in cases $R_{max}F_{1k}$ and $R_{max}F_{max}$, the measured latency slightly increases by 1 μs when the maximum number of flows is reached. The results show an identical scaling behavior for the Netronome SmartNIC when wildcard matching is used instead of exact matching. The slight increase in measured latency may be due to the increased lookup time required to access cached per-flow information in the Netronome SmartNIC.

The collected results from NetFPGA-SUME, shown in Fig. 3b, reveal that the measured latency is the same, with a median equal to 3.74 μs, for all the cases disregarding the number of installed rules or incoming flows. This latency slightly increases, around 0.1 μs, when wildcard matching is used. This shows that the NetFPGA-SUME is well designed to scale with traffic without compromising performance.

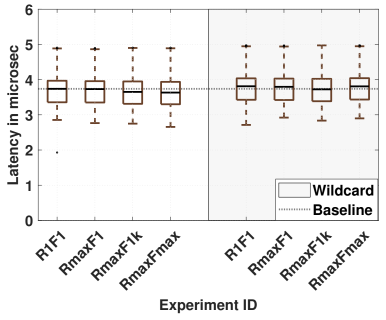
Unlike the latter two hardware-based P4 devices, the measured latency of the T4P4S software-based switch, presented in Fig. 3c, reveals a weak scaling behavior. When the number of installed rules increases in $R_{max}F_1$, the median of the measured latency stays constant at 45.4 μs. However, as the number of incoming flows increases in $R_{max}F_{1k}$ and $R_{max}F_{max}$ cases, the measured latency sharply increases to 75 μs. The same behavior is observed in the wildcard matching cases. The outliers with high values recorded in the case of T4P4S may be caused by batch processing taking place in the DPDK-backend of this device. Note that in this case, the maximum number of flows and rules is equal to 1k as illustrated in Table II, which makes the two cases $R_{max}F_{1k}$ and $R_{max}F_{max}$ equivalent.

Fig. 4 shows the average measured latency, in μs and logarithmic scale, for all the devices and all the considered flow scalability experiments with packet sizes equal to 256, 1000, and 1500 Bytes. In general, we can again observe that NetFPGA-SUME has a very stable performance when processing scaled traffic, followed by Netronome SmartNIC, then by T4P4S software switch, which has a very weak scaling behavior. The reason behind this is that the first two devices are hardware devices that are engineered to support worst-case traffic at line rate, unlike T4P4S, which is a software-based switch and can hit different memory bottlenecks when doing extensive lookup operations.

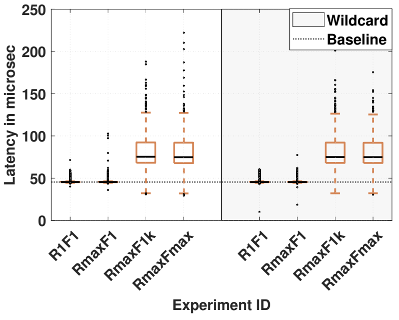
We can also observe that changing packet sizes have the effect of shifting the curves in the cases of the NetFPGA-SUME and the T4P4S software switch, where larger packet sizes lead to longer processing latency. However, in the case



(a) Netronome SmartNIC.

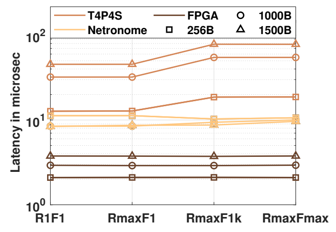


(b) NetFPGA-SUME.

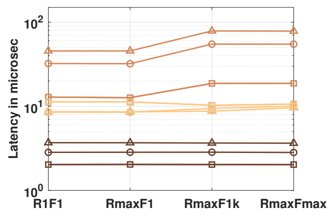


(c) T4P4S Switch.

Fig. 3: Forwarding latency of different flow scalability experiments for 1500 Bytes packets.

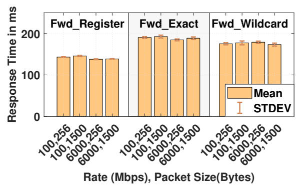


(a) Exact Matching.

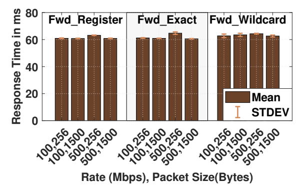


(b) Wildcard Matching.

Fig. 4: Average forwarding latency of different flow scalability experiments for different targets and packet sizes.



(a) Netronome SmartNIC.



(b) NetFPGA-SUME.

Fig. 5: Response time for different traffic loads and control plane rules to be updated.

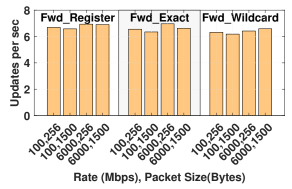
of the Netronome SmartNIC, we observe that smaller packet sizes, i.e., higher packet rates, lead to higher packet processing latency, due to the frequent memory operations required at high packet rates. Moreover, while scaling the number of incoming flows in the case of the Netronome SmartNIC slightly increases the latency when packet size is equal to 1000 and 1500 Bytes, it results in a slight latency decrease when packet size is set to 256 Bytes. This behavior may be due to optimization techniques implemented in the device, which are triggered in response to extensive lookup operations as in the case when the packet rate and the number of incoming flows are both high. The dependency of the latency on the packet size is due to the fact that P4 programmable devices behave as store-and-forward devices where they have to store the payload until the desired packet processing on headers takes place. Accordingly, the latency varies based on the size of the packet's payload.

B. Rule Update Responsiveness

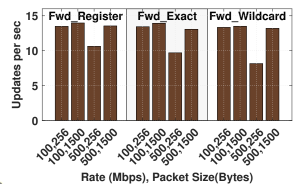
The results of the rule update responsiveness experiments are presented in this subsection. Fig. 5 shows the mean and standard deviation of the measured response times in μs when devices are under different traffic loads and dealing with different types of control plane update rules. Based on Fig. 5a, the average response time for updating registers in the case of Netronome SmartNIC ranges between 137 to 145 ms when different traffic loads are generated. This range increases to 185-192 ms when tables are updated according to exact rules, and to 173-179 ms when wildcard rules are used. Unlike the Netronome SmartNIC case, the response time of NetFPGA-SUME is invariant when the type of loaded rules varies. The response time always ranges between 61 and 65 ms.

We can observe from all these cases that the effect of varying the rates and packet sizes is minimal and does not exceed 10 ms. Note that the same holds when rate and packet size are set to 250 Mbps and 1000 Bytes respectively, which is the reason why we omit plotting the results for these cases. The standard deviation of the measured response time across the 20 trials for every case is always smaller than 3 ms in the case of NetFPGA-SUME and 10 ms in the case of Netronome SmartNIC. It is important to highlight that a major component contributing to this latency is the pre-processing delay taking place in the control plane tools used to communicate the update rule commands. This delay is measured in the case of NetFPGA-SUME where it reaches up to 60 ms.

Fig. 6 shows the results corresponding to the average update rate that Netronome SmartNIC and NetFPGA-SUME can support as a function of different traffic loads and types of



(a) Netronome SmartNIC.



(b) NetFPGA-SUME.

Fig. 6: Update rate for different traffic loads and control plane rules to be updated.

control plane rules. The update rate corresponding to the Netronome SmartNIC, shown in Fig. 6a, ranges between 6.2 and 7 updates per second independent from rate, packet size, or type of rules. This update rate increases in the case of NetFPGA-SUME, shown in Fig. 6b, reaching up to 14 updates per second. In this case, the update rate does not vary based on the type of reconfiguration command or traffic load except when the rate and packet size pair is equal to 500 Mbps and 256 Bytes. This is the maximum packet rate case (highest rate and smallest packet size), which results in a processing bottleneck in this card, likely in the interaction with the PCIe bus, leading to a low update rate that can drop down to 8 updates per second.

Note that in some of the test cases we removed a few outliers from the measured data points which were much larger than the rest of the measurements and can be classified as measurement noise. The total number of removed data points is less than 2.5 % of the overall collected measurement results.

IV. Related Work

Currently, few papers consider benchmarking the performance of P4 devices. Table IV summarizes and compares the performance of different P4 devices based on different evaluated criteria. The selected devices cover different types of processing platforms: CPU, NPU, FPGA, and ASIC [11]. The results of the first five evaluated criteria are surveyed from [5], [6], [12], [13], while the last two are added based on the evaluation conducted in this paper to create a full picture of the potentials and limitations of different P4 devices.

We can observe from the table that the hardware-software trade-off between performance and flexibility is dominant. When the packet processing device is more like an ASIC device, the performance in terms of throughput, latency, jitter, flow scalability, and responsiveness to rule updates prevail compared to that of software-like devices. However, this comes at the cost of reduced flexibility in defining new functionalities (P4 externs) on these devices, a limited capacity of resources that can be used, and a higher price. Note that we consider that CPU-based P4 devices are the cheapest as they can co-locate with other applications when running as software instances.

V. Conclusion

In this work, we benchmark the performance of different state-of-the-art P4 programmable devices, each belonging to a distinct category of processing platforms. The experiments are carefully designed and analyzed to determine the effect of different parameters that can influence the flow scalability and rule update responsiveness of the investigated devices. Results show that NetFPGA-SUME and Netronome SmartNIC can handle an increased number of flows without compromising packet processing latency, unlike T4P4S software-based switch. On the other hand, the rule update responsiveness analysis revealed that devices respond differently to control plane commands, but the response time is always in milliseconds range, i.e three orders of magnitude larger than data plane processing. Additionally, we observe that the pre-processing of control plane commands taking place in the device's provided toolchains contributes to a large part of the device's response time, which highlights the importance of optimizing the design of these toolchains.

This work can further be extended by performing a similar evaluation on other P4 programmable devices such as ASIC-based ones. In general, this evaluation highlights the importance of efficiently designing P4 programmable devices to unleash their potentials in shaping the future of networks.

TABLE IV: Comparing different P4 devices, belonging to different processing platforms, based on various criteria [5], [6], [12], [13]; (.) is used when information is not available.

P4 Device	**T4P4S**	**SmartNIC**	**NetFPGA**	**Tofino**
Criteria	*CPU*	*NPU*	*FPGA*	*ASIC*
Throughput	+	++	+++	++++
Latency	+	++	+++	++++
Jitter	+	++	+++	++++
Resources	++++	+++	++	+
Flexibility	++++	+++	++	+
Price	++++	+++	++	+
Flow Scalability	+	++	+++	.
Rule Update Response	.	++	+++	.

References

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[2] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki, "T4p4s: A target-independent compiler for protocol-independent packet processors," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2018, pp. 1–8.

[3] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "Netfpga sume: Toward 100 gbps as research commodity," *IEEE micro*, vol. 34, no. 5, pp. 32–41, 2014.

[4] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of the Internet Measurement Conference (IMC)*. ACM, 2015, pp. 275–287.

[5] H. Harkous, M. Jarschel, M. He, R. Priest, and W. Kellerer, "Towards understanding the performance of p4 programmable hardware," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2019, pp. 1–6.

[6] H. Harkous, M. Jarschel, M. He, R. Pries, and W. Kellerer, "P8: P4 with predictable packet processing performance," *IEEE Transactions on Network and Service Management*, 2020.

[7] Netronome. (2017) Netronome smartnic. https://www.netronome.com/products/smartnic/overview/. Accessed: 2021-04-29.

[8] (2017) Simple Sume Switch Architecture. https://github.com/NetFPGA/P4-NetFPGA-public/wiki/Workflow-Overview. Accessed: 2021-04-29.

[9] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki. (2017) T4P4S repository. https://github.com/P4ELTE/t4p4s. Accessed: 2020-12-18.

[10] M. Jacobsen, D. Richmond, M. Hogains, and R. Kastner, "Riffa 2.1: A reusable integration framework for fpga accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 4, Sep. 2015.

[11] Intel. (2021) Intel Programmable Ethernet Switch. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html. Accessed: 2021-04-29.

[12] D. Scholz, H. Stubbe, S. Gallenmüller, and G. Carle, "Key Properties of Programmable Data Plane Targets," in *Teletraffic Congress (ITC 32), 2020 32nd International*, Osaka, Japan, 2020.

[13] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon, "Whippersnapper: A p4 language benchmark suite," in *Proceedings of the Symposium on SDN Research*, 2017, pp. 95–101.