# Towards Understanding the Performance of Traffic Policing in Programmable Hardware Switches

Nemanja Đerić, Amir Varasteh, Amaury Van Bemten, Carmen Mas-Machuca, Wolfgang Kellerer
Chair of Communication Networks, Department of Electrical and Computer Engineering,
Technical University of Munich, Germany
Email:{nemanja.deric, amir.varasteh, amaury.van-bemten, cmas, wolfgang.kellerer}@tum.de

***Abstract*—To provide the predictability required by emerging applications, operators typically rely on policing and/or shaping at the edge to ensure that tenants do not use excess bandwidth that was not accounted for. One of the promises of 6G is to deploy applications with strict predictability requirements across subnets and even over the Internet, where policing cannot be implemented in the end hosts. This paper presents an empirical study of the ability of modern programmable network devices to implement predictable traffic policing in the network. We find out that none of the five investigated hardware switches can provide accurate traffic policing, a key requirement for providing predictable service to applications. We observe that the switches let applications send more than what they should be allowed to, reaching up to 60% and 100% relative error for the rate and burst parameters. We further uncover the fact that switches cannot police arbitrarily low bursts, e.g., not less than 13 kilobit for one of our switches. We investigate how such limitations impact the performance of state-of-the-art solutions for predictable latency such as *Chameleon*. We observe that, for ensuring its predictable guarantees, *Chameleon* rejects around 50% of the tenants it could accommodate if switches were perfect, hence decreasing by the same ratio the revenue for the operator. Based on these observations, we discuss solutions toward more accurate and predictable policing in wide-area networks.**

***Index Terms*—traffic policing, traffic shaping, network predictability, network measurements, software defined networks**

## I. Introduction

For providing their services, modern applications, e.g., related to health, business, and entertainment, impose more and more requirements on the networking infrastructure. Generally, such applications require the network to provide predictability. Besides ensuring that the performance of their networking equipment is predictable, that requires operators to make sure that the traffic sent by their tenants is also predictable. Indeed, unexpected bursts of traffic entering the network consumes bandwidth, build up queues, and potentially leads to unexpected packet losses and hence degraded performance for other tenants. To avoid that, operators rely on policing and/or shaping of the traffic at the edge of the network, typically performed directly on the end hosts [1]–[4].

Nowadays, such applications are mostly deployed in data centers [5], industrial networks [6], or enterprise networks [7]. 6G now introduces the idea of deploying these applications over the global Internet. The high and global connectivity offered by the Internet, combined with predictability, enables an ever-growing plethora of global services like smart cities, smart grids, or telesurgery. This shift from local area networks to wide-area networks prevents operators from accessing end hosts, e.g., for traffic policing. As a result, a key enabler for predictable performance in 6G networks is the ability to accurately and predictably perform traffic policing in the network.

Yet, while traffic policing is a well-known feature of network switches, the extent to which state-of-the-art programmable switches perform this task accurately and predictably has not been investigated in the literature. Motivated by this observation, this paper presents an extensive measurement study of the policing performance of five OpenFlow switches from three different manufacturers. We investigate the processing time overhead induced by configuring policing on the switches and quantify the policing accuracy and predictability in terms of the burst and rate parameters typically used for modeling traffic patterns.

Our observations are rather negative: none of the investigated switches perform policing accurately. While policing seemingly does not impact the processing time of switches, we find that the policed traffic deviates by up to 100% in terms of allowed burst and 60% in terms of rate for some configurations. Even more concerning, we observe that switches do not support the configuration of arbitrary burst values. Some of our switches require a minimum burst size (e.g., 13 kbits for our *Pica8* switches), others do not support the configuration of a burst value and only perform rate-based policing.

With the assumption that these inaccuracies can be perfectly modeled, we investigate how these would impact the performance of state-of-the-art solutions for providing predictable latency in programmable networks. Using the open-sourced code of the *Chameleon* system [1], [8] for cloud networks, we quantify how much these inaccuracies impact the number of tenants the system can accommodate while still providing its predictability guarantees. Astonishingly, we observe that *Chameleon* reduces the number of tenants it accepts by around 50% compared to a situation where switches are deemed perfect. This means that the sole limitations of the policing feature of switches force operators to see their revenue halved. We accordingly discuss potential solutions and alternatives to circumvent the aforementioned limitations.

The rest of this paper is organized as follows. In Section II, we present a background on network predictability and justify the need for accurate traffic policing. In Section III, we in-

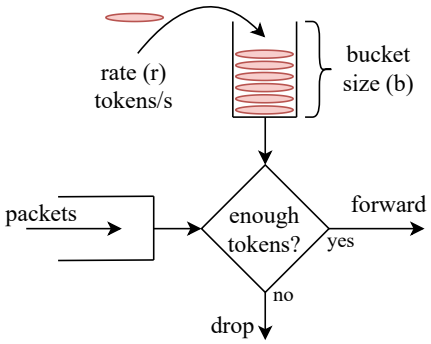



Fig. 1: Diagram representing token bucket algorithm.

troduce our measurement setup and procedure. After that, different measurement cases along with the results are presented in Section III-C, followed by a discussion in Section IV. The related work is summarized in Section V. Finally, Section VI concludes the paper.

## II. Background

In this section, we explain the importance of limiting the rate and burst of traffic flows in predictable networks (see Sec. II-A). Thereafter, we present in more detail how it is realized in OpenFlow networks (see Sec. II-B).

### A. The Need for Traffic Policing

Providing predictable latency in programmable networks requires the computation of performance bounds, in particular delay and throughput, in the network. Deterministic network calculus (DNC) [9], [10] is the main framework used by state-of-the-art solutions for predictable latency [1], [2], [5] to compute performance bounds.

DNC is a system theory for communication networks based on the min-plus algebra. Based on traffic and node models, DNC allows to derive *(i)* the maximum per-packet delay traffic can experience at a node, *(ii)* the maximum backlog (e.g., amount of data) traffic will generate at a node, and *(iii)* the updated traffic model at the output of the node. Altogether, these bounds enable operators to provide guarantees to their tenants.

Inevitably, a key requirement for the bounds computed by DNC to be valid is for the traffic and node models to be correct worst-case models. In DNC terminology, the traffic model is referred to as the *arrival curve* and the node model is referred to as the *service curve*. Many works have focused on determining the service curve of network nodes, e.g., the recent *Loko* system [2]. Given a service curve, it is crucial to ensure that the traffic entering the node is not exceeding its arrival curve. Indeed, if the traffic violates the arrival curve used to compute delay and performance bounds, all the guarantees provided to the applications would vanish. For example, slightly exceeding the expected traffic envelope can increase buffer occupancy at some nodes, thereby potentially reaching the buffer capacity of a node, and hence generating packet loss, retransmission, and degraded performance.

Tenants requesting predictable performance from the network are expected to provide the arrival curve of the traffic they wish to send in the network. Because tenants cannot be trusted, operators must ensure that the traffic sent by the different tenants does not exceed the agreed arrival curve. This can be achieved by adding a DNC processing element before the traffic enters the network and that ensures that its output traffic respects the agreed arrival curve (see *(iii)* above). This can be done by either delaying or dropping packets that would lead to the arrival curve being violated. This is called *shaping* and *policing*, respectively. We here focus on policing.

### B. Traffic Policing in OpenFlow

The *metering* feature of OpenFlow switches [11] allows to police traffic and ensure that it respects an agreed arrival curve. Metering is introduced in OpenFlow v1.3 [11] and it is widely supported by carrier-grade switches. A (hardware) entity that performs the policing is called a meter. In OpenFlow, meters and flow rules are disjoint. Hence, to police a traffic flow, a meter has to be configured and assigned to the corresponding flow rule as part of the instructions of the flow rule.

The main type of arrival curve used in the literature is a token bucket arrival curve. Traffic is modeled using two parameters: allowed burst and allowed rate. This corresponds to the leaky token bucket algorithm (see illustration on Fig. 1) [12]. Initially, the token bucket contains $b$ tokens (i.e., bits or packets) which determines the maximum burst size of a flow. The bucket is continuously filled with $r$ tokens per second, which defines the rate of a flow. When an $N$-bit packet arrives, the number of available tokens in the bucket is checked. If this value is higher than $N$, the packet is forwarded, and the $N$ tokens are removed from the bucket. If there are not enough tokens available (i.e., value is lower than $N$), the packet is either dropped or it is remarked[1] and forwarded.

Various metering configurations are supported [11], such as: *i)* rate policing (either kbps or packets per second (pps)), *ii)* burst size policing (either kbps or pps), *iii)* different band types (excess traffic is either dropped or remarked and forwarded), *iv)* collecting metering statistics.

The main challenge here is to investigate if the traffic policing functionality of modern carrier-grade switch matches the theoretical properties of the leaky token bucket algorithm. In particular, we focus on the following research questions:

1) How can we properly measure the performance of a traffic policer?
2) How accurate is the traffic policing feature of carrier-grade switches?
3) What is the impact of potential traffic policing inaccuracies on the performance of systems providing predictable latency?

## III. Measurements

In this section, we firstly introduce the measurement setup and procedure (see Sec. III-A). Then we discuss how the rate

[1]Packet (priority) remarking refers to lowering the drop precedence of the DSCP field in IP header.

TABLE I: Specifications of the evaluated switches: names, ASIC, CPU, and ports.

| Switch | ASIC | CPU | Ports |
|---|---|---|---|
| *HP E3800* | HPE ProVision | Freescale P2020 | 48×1G-RJ45 + 4×10G-SFP+ |
| *DELL S3048-ON* | Broadcom StrataXGS | *undisclosed* | 48×1G-RJ45 + 4×10G-SFP+ |
| *DELL S4048-ON* | *undisclosed* | *undisclosed* | 48×10G-SFP+ + 6×40G-QSFP+ |
| *Pica8 P3290* | Broadcom Firebolt 3 | Freescale MPC8541CDS | 48×1G-RJ45 + 4×10G-SFP+ |
| *Pica8 P3297* | Broadcom Triumph 2 | Freescale P2020 | 48×1G-RJ45 + 4×10G-SFP+ |

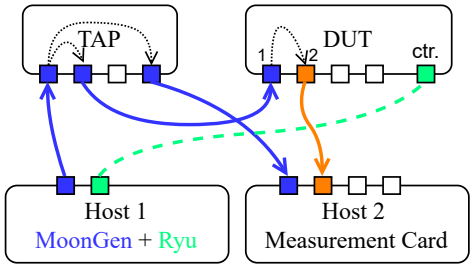



Fig. 2: The measurement setup.

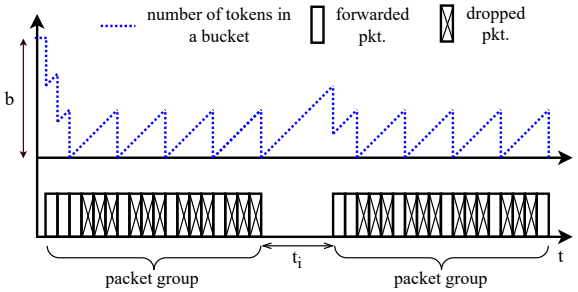



Fig. 3: An illustration of one time-wise sequence of pre-policed and policed packets with the corresponding token bucket state. The scenario depicts two packet groups, where each one contains at least 3 packets that are not part of the initial burst.

and burst size can be derived from the measurement traces by utilizing the concept proposed in the DNC framework (see Sec. III-B). The measurement results are presented in Sec. III-C. In particular, we analyze what policing functionalities do the considered switches support, and how many meters they provide (see Sec. III-C1). Afterward, we discuss the impact of policing on the packet processing time (see Sec. III-C2). At the end, the accuracy of policing in terms of rate (see Sec. III-C4) and burst (see Sec. III-C5) is investigated.

### A. Setup

In this section, we introduce the measurement setup, parameters, and procedure. Table I shows the OpenFlow switches that we benchmark. To measure the performance of the traffic policing feature of these switches, we construct the following measurement setup presented in Fig. 2. In total, we use two servers, one networking tap, and a device under test (DUT), i.e., one of the switches from Table I. On the first server (i.e., Host 1 in Fig. 2), a Ryu SDN controller [13] is deployed, running a custom application that configures the DUT with a certain number of flow rules, and police the traffic based on the parameters listed in Table II. We note that the parameter values are chosen based on the related *state-of-the-art* approaches [1], [5]. Configuring a DUT (i.e., modifying flow rules and meters) is done through OpenFlow 1.3 (green dashed line in Fig. 2). Each inserted flow rule matches the incoming traffic with a certain unique IP address on port 1. The matched traffic then passes through exactly one unique meter, and it is forwarded on port 2 (packets can be dropped depending on the policing outcome). Additionally, Host 1 runs Moongen traffic generator [14] to send the data plane pre-policed traffic (blue line in Fig. 2). This generated traffic is mirrored by the networking tap device, and it is forwarded to both, the DUT and the second server which contains Endace DAG 7.5G4 measurement card [15]. Finally, based on the configured rules, the DUT forwards the traffic to the Host 2 server (equipped with a measurement card). Thus, we can obtain traffic traces of both, pre-policed and policed traffic on the second server. We do not consider cases if the total pre-policed rate is lower or equal to the total configured policing rate.

*Traffic Generation.* Fig. 3 illustrates an exemplary time series of the generated pre-policed traffic during one measurement run. *Host 1* generates multiple randomly-spaced groups of packets at line rate. The idea behind sending a group of packets at line rate is two-fold. Firstly, we aim to empty the token bucket of the corresponding meter to observe the accuracy of burst policing. Secondly, we strive to observe the accuracy of the token generation rate after the bucket is emptied. To achieve these goals, considering the token bucket rate and burst, the length of the generated packet group has to be big enough to observe a burst and a few packets (10 packets in our case) policed at line rate.

Further, to explore how the policing behaves with different initial states of token bucket (i.e., number of tokens), we vary the time between packet groups $t_i$ with a uniform distribution:

$$t_i(r, b) = U(0, 1.5 \times \frac{b}{r}), \tag{1}$$

where $\frac{b}{r}$ is the time needed to completely fill the token bucket of a meter. We multiply it by 1.5 to make sure that the token bucket can be filled fully again during one measurement run.[2]

[2]If multiple meters are used during the same run, the corresponding generated packet groups (one group belongs to one meter) are interleaved. In these cases, $t_i$ (Eq. 1) separates interleaved packet groups.

TABLE II: Considered Parameters

| Parameter | Abbreviation | Configured on | Values |
|---|---|---|---|
| *Number of flows* | $n$ | Ryu, MoonGen | 1, 10 |
| *Policing rate [kbps]* | $r$ | Ryu, MoonGen | 10, $10^2$, $10^3$, $10^4$, $10^5$ |
| *Burst size [kbits]* | $b$ | Ryu, MoonGen | 13, 15, 30, 50, 75, 100, 200, 300, 500, 1000 |
| *Packet size [byte]* | $s$ | MoonGen | 100, 500, 1000, 1500 |

### B. Deriving Rate and Burst

In this subsection, we explain how the rate and burst size of a policed flow are derived from the measurements. To derive the valid rate and burst values, we rely on token bucket properties and the methodologies provided in the DNC framework. In DNC, the token bucket algorithm (see Sec. II-B) constrains the burst $b$ and rate $r$ of a flow with the token bucket arrival curve [9], [10], defined as $\gamma_{r,b}$:

$$\gamma_{r,b} = \begin{cases} b + r \times t & , \forall t > 0 \\ 0 & , otherwise \end{cases}, \tag{2}$$

where $t$ is time.

To derive a valid (constraining) token bucket curve (rate $r$ and burst $b$) from the trace, we start with applying min-plus self-deconvolution [9], [10] on the measured policed traffic flow to produce its minimum arrival curve [9], [10]. In particular, the minimum arrival curve represents a valid flow model that can be used as an input parameter (flow description) for providing guarantees with DNC. For a detailed explanation of the DNC framework, we refer the readers to [9].

There are many ways of deriving a valid (constraining) token bucket arrival curve from the minimum arrival curve. Any curve which is above the minimum one represents a valid solution. In this paper, we first derive the rate of a flow from a minimal arrival curve. Afterward, we derive the burst size based on the calculated rate.

*Rate.* To derive the rate, we take the timestamp of the first ($t_a$) packet which is not part of the initial part ($t < t_a$) of the minimum arrival curve (see Fig. 4)[3]. In our scenario, we can find the first packet based on the packet inter-arrival times. We define rate $r$ as the maximal slope of the minimum arrival curve $y$ between $t_a$ and any other time instance $t \geq t_a$ (e.g., $t_b$ in Fig. 4):

$$r = \max_{\forall t, t > t_a} \left( \frac{y(t) - y(t_a)}{t - t_a} \right). \tag{3}$$

*Burst.* To calculate the burst size, we simply find the minimal $b$ which satisfies the following equation:

$$b + r \times t \geq y(t). \tag{4}$$

Using the previously explained procedure, we ensure that the derived token bucket arrival curve $\gamma_{r,b}$ is always above the self-deconvoluted minimum arrival curve. Hence, it is constraining the flow with rate $r$ and burst $b$.

[3]At higher policing rates, switches forward the packets in microbursts (in line rate). In such cases, we take the timestamp of the last packet of the first microburst. This effect is presented in Sec. III-C6.

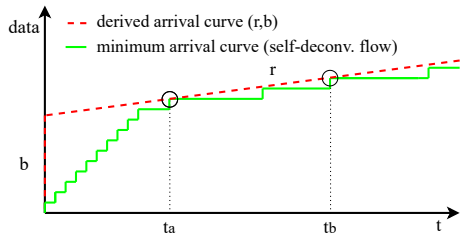



Fig. 4: An illustration of one time-wise sequence of policed packets (measured traffic trace) with the corresponding self-deconvoluted function and the constraining token bucket arrival curve.

### C. Results

*1) Policing Flags Support:* Table III lists the supported policing flags and the total number of available meters within each switch. Furthermore, for easier comparison, the table lists their corresponding flow table size (values are taken from our previous work [16]). To begin with, we observe that all the switches do support a traffic policing feature, except the ones manufactured by Dell (i.e., 1G S3048-ON and 10G S4048-ON). Therefore, we only consider *Pica8* and HP switches for the measurements. Moreover, considered HP device only supports rate policing, while *Pica8* switches support also burst policing in addition to rate. As a result, this limitation significantly constrains the usage of HP switches in predictable networks. This is discussed in more details in Sec. IV.

As it can be seen in Table III, *Pica8* switches support significantly more meters compared to HP. Since *state-of-the-art* approaches rely on a fine-grained flow control [6], the total number of flows on each switch in the network can easily grow up to several thousands [1] (e.g., over 2000 flow rules). Thus, having around 2000 meters may be insufficient for some use-cases. Furthermore, *Pica8* switches are equipped with more meters than the flow table size. This can facilitate the deployment of hierarchical traffic policing approaches [17].

Finally, regarding the band types (see Sec. II-B), it is observed that HP and *Pica8* switches support both drop and packet remarking.

*2) Processing Time:* To measure the impact of traffic policing on the packet processing time of each switch, we perform two sets of measurements, with and without policing. To disable policing, we do not configure the flow rules to forward the matched traffic to each corresponding meter. The processing time of each packet is the difference of a packet's timestamp before and after policing it. In order to identify

TABLE III: Supported Policing Flags, and Number of Meters

| Switch-Type | Policing Flags | | | | Band Type | | Number of | |
|---|---|---|---|---|---|---|---|---|
| | rate | pps | burst | stats | drop | dscp_remark | meters | flows |
| *HP E3800* | ✓ | ✓ | × | ✓ | ✓ | ✓ | 2047 | ca. 4085 |
| *Pica8 P3290* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 4096 | 2046 |
| *Pica8 P3297* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 8192 | 4094 |
| *DELL S3048-ON* | × | × | × | × | × | × | -/0 | 1000 |
| *DELL S4048-ON* | × | × | × | × | × | × | -/0 | 1000 |

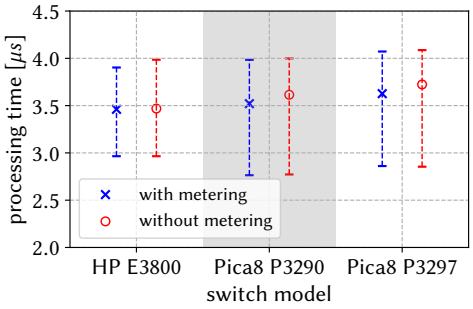



Fig. 5: Impact of metering on the processing time. All the measurement data is aggregated.

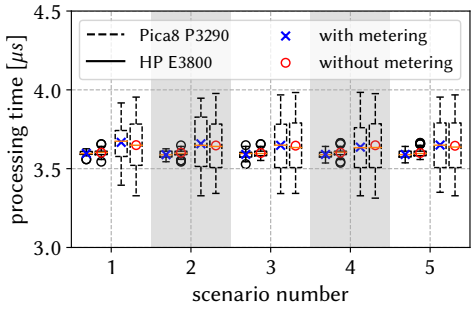



Fig. 6: Impact of metering on the processing time for five different measurement scenarios.

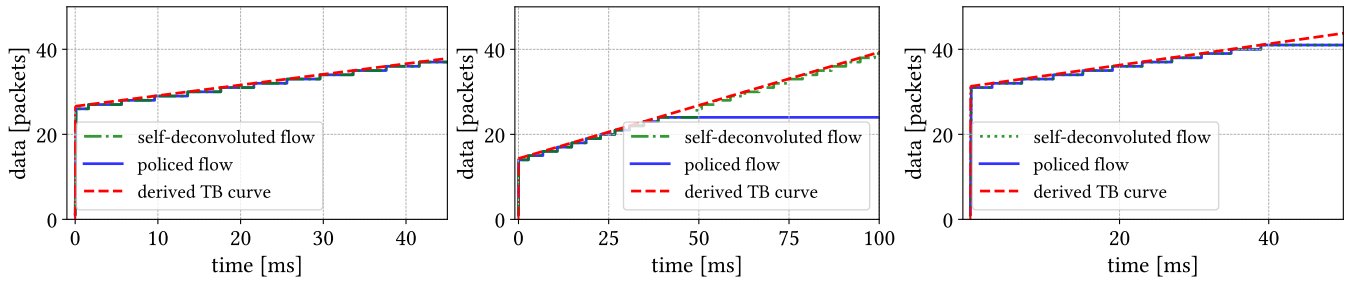
each packet, each packet has a unique MAC address.

*Outcome.* The minimal, maximal, and average packet processing time of switches are presented in Fig. 5 (the data is consolidated and based on all the measurement runs). Overall, there is no statistically significant impact of policing on the processing time. For instance, the minimal (and maximal) packet processing time of *HP 3800* is the same in both cases, i.e., $t_{min} \approx 3\ \mu s$ (and $t_{max} \approx 4\ \mu s$). To be more clear, we also illustrate five specific scenarios in an isolated manner in Fig. 6. These scenarios have the same packet size of 1000 bytes, but different configured policing rate and burst size (see Fig. 6). Accordingly, Fig. 6 shows that first, the processing time of the *Pica8* switch is higher than HP. More importantly, it can be seen that the packet processing time with and without policing is almost the same, even for different policing rates. Thus, it can be concluded that the policing is implemented in hardware (processing time usually varies a lot in software implementations [16]). Furthermore, it can be assumed that all the results presented in comprehensive *state-of-the-art* hardware measurement studies [16], [18], [19] can be valid in cases with policing enabled.

*3) Verifying the Derived Token Bucket Curves:* Fig. 7 depicts the initial part of the policed traffic trace, its min-plus self-deconvolution (i.e., minimum arrival curve), and the derived token bucket arrival curve for three different measurement runs. Since the switches police the traffic according to the token bucket algorithm, the initial parts of the self-deconvoluted function and policed traffic trace are very similar (see Fig. 7). Furthermore, the derived rate and burst of each token bucket arrival curve indeed fits well to the corresponding minimum arrival curve. We note that relying on the configuration values for generating a valid arrival curve has never been sufficient in our experiments. This means that the evaluated switches have always forwarded more traffic than expected.

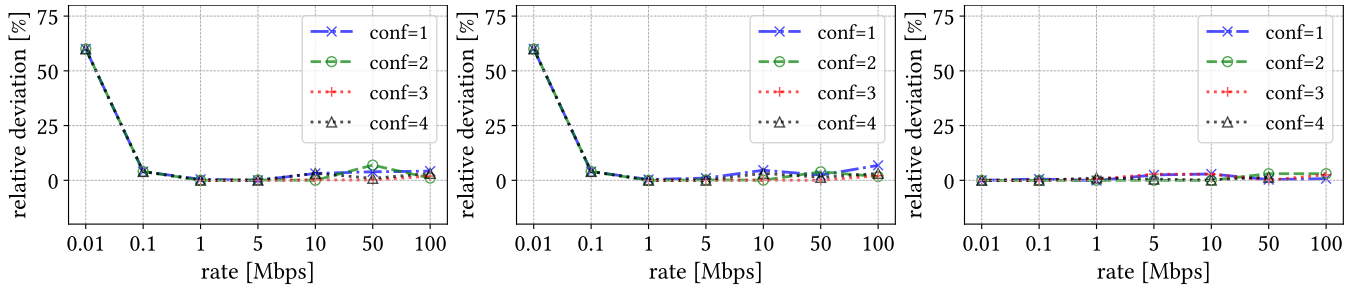
*4) Rate Deviation:* Fig. 8 illustrates the relative deviation of the derived policing rate from the configured one for *Pica8 P3290*, *Pica8 P3297*, and *HP E3800*. Overall, the derived rate often exceeds the configured one (average deviations is around $\sim$+1%), hence, the switches can actually forward excess traffic into the network. This can lead to delay violations and even packet loss. Furthermore, in the case of lower rates, both *Pica8* switches significantly deviate from the expected flow rate. For instance, if we configure a meter to police the traffic with the rate of 10 kbps, the forwarded policed rate is around 60% higher than expected, i.e., $\sim$ 16 kbps. This unexpected inaccuracy can have a significant impact on deploying *state-of-the-art* solutions, which is further investigated in the discussion section. Nevertheless, these deviations appear to be predictable, since, in different scenarios with the same configured/expected rate, the switches police the traffic with similar accuracy. In fact, it indicates that these switches can be used in predictable networks (if the error is accounted for).

*5) Burst Deviation:* To study the burst deviation, we start with Fig. 9a which presents the relative deviation of the derived burst size from the configured one for *Pica8 P3297* switch. In this case, we consider four different scenarios with the same parameters except for the packet size (i.e., number of flows is 1 and the configured rate is 1 Mbit). Firstly, it can be observed that the derived burst size is always higher than the configured/expected one, which can be detrimental in predictable networks. Also, the relative burst deviation depends on the configured one. For the higher values, the inaccuracy is usually below 5%. For example, the relative




(a) Pica8 P3290 (b) Pica8 P3297 (c) HP E3800

Fig. 7: An initial part of the policed traffic trace, its min-plus self-deconvolution, and the derived token bucket arrival curve for the considered switches. In all three scenarios, policing rate is 1 Mbps, and the packet size is 500 bytes. The configured burst size for *Pica8 P-3290* is 100 kbits (or 25 500 byte packets) and for *Pica8 P-3297* is 50 kbps (approx 12.5 500 byte packets).




(a) Pica8 P3297. (b) Pica8 P3290. (c) HP E3800.

Fig. 8: Achieved accuracy (in terms of relative deviation) of rate policing depending on the configured policing rate. We consider four different measurement configurations with the following parameters, $c1 \rightarrow (b = 30$ kbps, $s = 100$ bytes, $n = 1)$; $c2 \rightarrow (b = 100$ kbps, $s = 500$ bytes, $n = 1)$; $c3 \rightarrow (b = 200$ kbps, $s = 1500$ bytes, $n = 1)$; $c4 \rightarrow (b = 75$ kbps, $s = 1000$ bytes, $n = 1)$. Since it is not possible to configure the burst size on *HP E3800*, the corresponding parameters is ignored.

error is always lower than 3% for the configured burst size of 500 kbits. Surprisingly, for configurations with smaller burst size, the relative error can even exceed 100%. Moreover, even for the same configured burst sizes, the relative burst deviation varies with the packet size. This indicates that it is not possible to fully compensate for these inaccuracies with precise modeling. Even if we perfectly model the error of a device, users might generate flows with dynamically varying packet sizes.

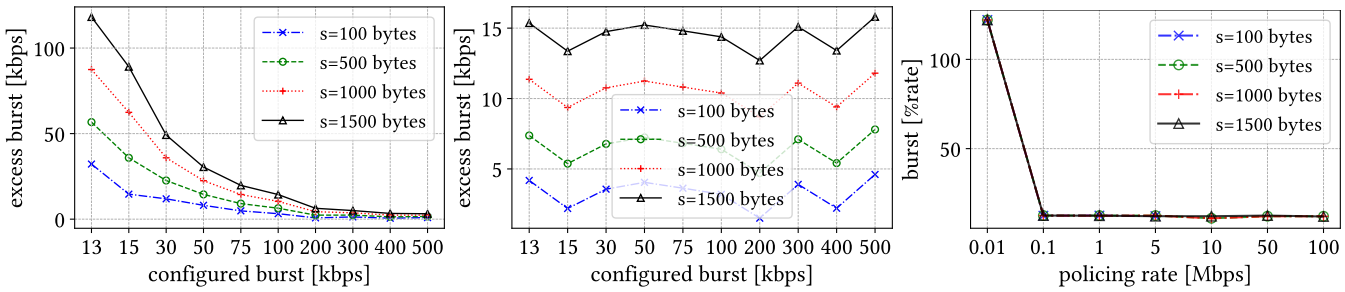
Further, we note that the absolute burst deviation from the configured value of *Pica8 P3297* does not depend significantly on the configured one (see Fig. 9b). In fact, the absolute burst deviation is always between 0 and 16 kbits. This suggests that each meter introduces a similar amount of excess traffic in the network, regardless of the configuration. Additionally, it is not possible to configure a meter of *Pica8 P3297* with a burst value lower than 13 kbits.

The results for *Pica8 P3290* follow the same trend, thus we omit showing them.

Regarding the *HP E3800* switch, since it does not support configuration of the burst size, we present the results without varying this parameter. Fig. 9c presents the derived burst size for four different scenarios with varying policing rate. Even though we cannot configure the burst size of HP, it can be observed that the burst size is predictable (see Fig. 9c). That is, runs with different measurement parameters produce almost identical results. Furthermore, the results show that the burst size is correlated with the configured policing rate (see Fig. 9c). For instance, if the configured/expected rate is lower than 100 kbps, the measured burst size is slightly bigger (i.e., $\sim 12.5$ kbits) than the maximal size of an ethernet packet (i.e., $1500 \times 8$ bits $= 12$ kbits)[4]. For the higher rates, the burst size corresponds to $\sim 12.5\%$ of the configured rate.

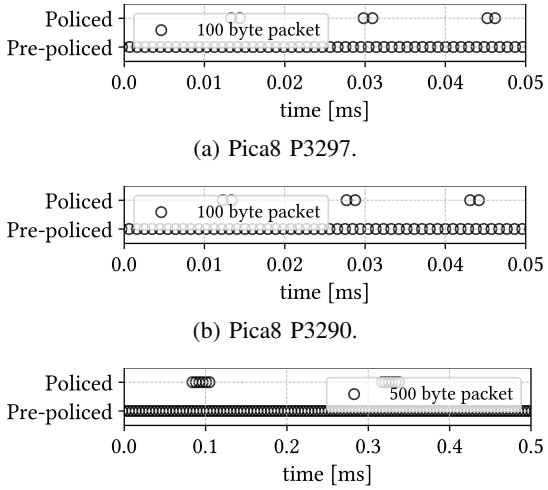
*6) Microbursts at High Policing Rate:* Fig. 10 illustrates three very short time snippets taken from different (high-rate) measurement runs for *Pica8 P3297*, *Pica8 P3290*, and *HP E3800*. The time snippets show the relative timestamps of pre-policed and policed traffic (i.e., packets) shortly after the token bucket is emptied (with an initial burst). In theory, the token bucket algorithm generates tokens continuously. Therefore, since the pre-policed traffic is sent at line rate, the outgoing policed traffic in the depicted scenarios should be uniformly spaced. For example, in the case of *Pica8 P3297*

[4]This is probably done in order to accommodate adding multiple VLAN tags (one tag is 4 bytes).




(a) Pica8 P3297 - relative error. (b) Pica8 P3297 - excess burst. (c) HP-3800.

Fig. 9: Derived burst size from the measurements (with different packet sizes $s$) for *Pica8 P3297* and *HP E3800*. The derived rate for *Pica8 P3297* in all presented scenarios deviates at most $0.5\%$ from the configured value.




(a) Pica8 P3297.

(b) Pica8 P3290.

(c) HP E3800.

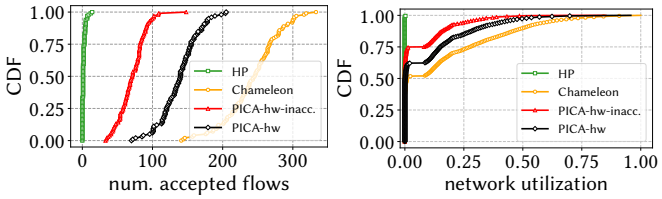
Fig. 10: Microbursts generated by *Pica8* and HP devices at higher policing rates (i.e., $r = 100$ Mbps) with smaller packet sizes (e.g., $s \leq 500$ bytes).

(see Fig. 10a), the time to generate enough tokens for one packet is 8 $\mu s$ (policing rate is 100 Mbps and packet size is 100 bytes). Hence, an ideal token bucket algorithm should forward six uniformly separated packets every 8 $\mu s$. However, in practice, we observe that this is not the case. The packets are actually forwarded in small (micro) bursts (e.g., two packets for *Pica8 P3297*, see Fig. 10a). In fact, all the considered switches exhibit the same behavior (see Fig. 10). However, this effect is only observable at high policing rates (e.g., $r \geq 50$ Mbps) with small packet sizes (e.g., $s \leq 500$ bytes). Therefore, we can suppose that the token bucket is discretely filled with a time interval that can be greater than the time needed to generate enough tokens for a packet. This effect can be accounted for by increasing the derived burst size accordingly.

## IV. Discussion

In the previous section, it is shown that even the carrier-grade switches suffer from some hardware limitations, and inaccurate traffic policing. Depending on the brand and model of the switch, these inaccuracies can occur for rate and/or burst policing. Two approaches can be followed to resolve these policing issues. Firstly, the hardware implementation of the switches can be further improved. Secondly, the policing inaccuracies could be modeled and accounted for in the network management frameworks. In the second case, accounting for these policing inaccuracies can lead to lower performance of such networks (e.g., lower network utilization). To discover the impact of these policing limitations on the network performance, we use the simulation tool from our previous work, *Chameleon* [1], a cloud provider system which delivers strict end-to-end delay guarantees to network flows. In *Chameleon*, a flow is characterized by a source-destination node pair, data rate, burst size, and the required delay. *Chameleon* relies on DNC theory, priority queuing, and a simple greedy algorithm to provide the delay-constrained path allocation to the incoming traffic flows, with no packet loss. In particular, given a network and a set of flows, we are interested to find out how the policing limitation of the switches can affect the utilization of the network. To do so, we use a 4-fat-tree network with 10 servers per rack, each hosting 10 virtual machines. The network switches are considered with four priority queues (each with different assigned delays), and 1 Gbps of link bandwidth. The network flows are generated randomly according to service types used in our previous work [1], normally distributed. To show the impact of the hardware limitation, we compare four cases:

1) *Chameleon*: The policing is set to be 100% accurate (the perfect case).
2) *PICA-hw*: *Pica8 P3297* switches are deployed in the whole network. In this case, we consider the hardware limitation by setting the minimum burst size to 17 kbits (based on the results in Section III-C5). However, we assume that policing inaccuracies are resolved.
3) *PICA-hw-inacc.*: *Pica8 P3297* switches are considered similar to the previous scenario. In addition to the hardware limitations (i.e., setting the minimum allowed burst size to 17 kbits), according to Fig. 9b, the effect of burst policing inaccuracy is included.
4) *HP*: The network switches are considered to be HP E3800, which does not support burst policing. In




(a) (b)

Fig. 11: The role of accurate traffic policing in (a) number of accepted flows, and (b) network utilization.

this case, we generate a model based on Fig. 9c which translates the expected burst size and rate to the deployed policed configurations of the HP switches.

Considering these four cases, we compare the total number of accepted flows and the achieved network utilization in Fig. 11. Surprisingly, Fig. 11a shows that the cost of hardware limitation and policing inaccuracy can be very high. In particular, compared to the perfect policed case (i.e., *Chameleon*), it can be seen that the network consisting of *Pica8* switches accepts around 50% less flows. Since the OpenFlow metering feature in *Pica8* switches cannot be configured with a lower burst size than 17 kbits, the flows with lower burst requirements are actually being policed with a higher burst. This leads to the waste of resources in the network, i.e., a lower number of accepted flows for *PICA-hw* case. Also, it can be seen that considering the burst policing inaccuracy (*PICA-hw-inacc.*) causes the network to accept even fewer flows than the *PICA-hw* case (overall around 35% of the perfect case). For the *HP* case, since it does not support configuring burst policing, the number of accepted flows is very low, around 2% of the perfect scenario. Similarly, the considered traffic policing limitations also decreased the network utilization significantly (see Fig. 11b). These shocking values indicate that traffic policing plays an important role in the performance of the networks.

## V. Related Work

Recent works in the literature have focused on improving the host-based traffic policing approaches [1], [4], [5]. Although these solutions can achieve high accuracy [4], they often cannot support high rates [1], especially if the packet size is small. Moreover, opposed to our work, these solutions are not applicable in scenarios where in-network traffic policing is needed.

The responsible unit to perform the in-network policing is the network switches. Generally, the measurement and investigation of different performance metrics of the OpenFlow switches have been receiving a lot of attention in the literature [2], [16], [18]–[26], [26]–[35].

On the one hand, some works have studied the mismatch between the control and data plane states of a device [16], [18], [20]–[23], [25]–[28]. As an interesting finding, authors in [16] have shown that inserting a new OpenFlow rule into the flow table of a carrier-grade OpenFlow hardware switch can take a significant amount of time (can be over a second). Worse than that, sometimes the control plane state claims that a forwarding rule is inserted in the switch, while it has never been inserted [16], [18], [21]. As we know, to police a traffic flow, the switch has to use a forwarding rule to match the traffic and forwards it to the corresponding meter. Therefore, these mismatching issues can affect the policing function as well and need to be accounted.

On the other hand, some works have focused on measuring the data plane performance of carrier-grade OpenFlow switches [2], [19], [26], [29]–[35]. For instance, [2], [16], [19] have measured the available hardware flow table size of various SDN-enabled switches and compared their performance. Durner. et al. [24] have measured and evaluated different Quality of Service (QoS) metrics in OpenFlow switches such as the priority queuing. However, to the best of our knowledge, we are the first work that investigates the performance of policing feature of the OpenFlow switches. Although the packet processing time of the switches has been studied in other works [16], [19], [26], [35], none of them have considered the impact of traffic policing. In addition to the processing time, we study the accuracy of the rate and burst policing on a set of carrier-grade OpenFlow switches.

## VI. Conclusion

This paper presented the first steps towards measuring and modeling the performance of in-network traffic policing in carrier-grade OpenFlow switches. Considering a set of modern carrier-grade OpenFlow switches, we have measured the impact of the policing on the packet processing time. Also, we have proposed a measurement methodology and presented results for determining the accuracy of rate and burst policing of network traffic flows. This methodology is generalizable and can be used for evaluating the policing performance of other networking switches, if they support traffic policing.

We found that not all the evaluated switches (e.g., Dell S4048-ON) support OpenFlow traffic policing feature. For those who support, the policing feature has almost no impact on the packet processing time, which is interesting from the predictability point of view. However, we have observed these switches have some policing limitations, e.g., it is not possible to have a burst lower than a certain value in HP and *Pica8* brands. As a result, these switches may not be suitable for use-cases such as industrial networks, where usually the burst is small [6]. Additionally, our measurements have shown that these switches do not perform the policing accurately, especially for traffic flows with low rate and burst size.

We performed a study to find out what is the impact of these limitations and inaccuracies in a realistic cloud network settings. To do so, we used the simulation tool from our previous work [1], which provides predictable latency guarantees using DNC, in a cloud data center network. The simulation results indicated that these limitations can actually have a significant impact on the network, especially in terms of the number of accepted flows and network utilization.

We see our work as a first step and believe that it opens several interesting avenues for future research. In particular, the burst and rate policing inaccuracies can be modeled and accounted for in the predictable network modeling frameworks, such as DNC. Moreover, it would be interesting to investigate if more accurate traffic policing can be realized with other programmable networking technologies, such as P4.

## ACKNOWLEDGEMENT

This work is partially funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of the project "5G Testbed Bayern mit Schwerpunktanwendung eHealth", and partially by the Germany Federal Ministry of Education and Research under project AI-NET ANTILLAS (project ID #16KIS1318).

## REFERENCES

[1] A. Van Bemten, N. Đerić, A. Varasteh, S. Schmid, C. Mas-Machuca, A. Blenk, and W. Kellerer, "Chameleon: predictable latency and high utilization with queue-aware and adaptive source routing," in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pp. 451–465, 2020.

[2] A. Van Bemten, N. Đerić, J. Zerwas, A. Blenk, S. Schmid, and W. Kellerer, "Loko: Predictable latency in small networks," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pp. 355–369, 2019.

[3] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can {JUMP} them!," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 1–14, 2015.

[4] A. Saeed, N. Dukkipati, V. Valancius, V. The Lam, C. Contavalli, and A. Vahdat, "Carousel: Scalable traffic shaping at end hosts," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 404–417, 2017.

[5] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable message latency in the cloud," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 435–448, 2015.

[6] J. W. Guck, A. Van Bemten, and W. Kellerer, "Detserv: Network models for real-time qos provisioning in sdn-based industrial environments," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1003–1017, 2017.

[7] C. Sieber, S. Schwarzmann, A. Blenk, T. Zinner, and W. Kellerer, "Scalable application-and user-aware resource allocation in enterprise networks using end-host pacing," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 5, no. 3, pp. 1–41, 2020.

[8] A. Van Bemten, "Chameleon: network controller." https://github.com/amovanb/chameleon-controller, 2021.

[9] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*, vol. 2050. Springer Science & Business Media, 2001.

[10] A. Van Bemten and W. Kellerer, "Network calculus: A comprehensive guide," 2016.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[12] D. C. Lee, "Effects of leaky bucket parameters on the average queueing delay: Worst case analysis," in *Proceedings of INFOCOM'94 Conference on Computer Communications*, pp. 482–489, IEEE, 1994.

[13] Ryu, SDN, "Framework community: Ryu SDN framework." http://osrg.github.io/ryu, 2015.

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

[15] E. T. Limited, "Endace DAG 7.5G4 datasheet"." https://www.endace.com/dag-7.5g4-datasheet.pdf, 2016. Accessed: 2018-10-26.

[16] A. Van Bemten, N. Đerić, A. Varasteh, A. Blenk, S. Schmid, and W. Kellerer, "Empirical predictability study of sdn switches," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 1–13, IEEE, 2019.

[17] P. Documentation, "Configuring metering in picaos." https://docs.pica8.com/display/PicOS374sp/Configuring+Meter, 2021. Accessed: 2021-1-17.

[18] M. Kuźniar, P. Perešíni, D. Kostić, and M. Canini, "Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches," *Computer Networks*, vol. 136, pp. 22–36, 2018.

[19] S. Bauer, D. Raumer, P. Emmerich, and G. Carle, "Behind the scenes: what device benchmarks can tell us," in *Proceedings of the Applied Networking Research Workshop*, pp. 58–65, 2018.

[20] M. Kuzniar, P. Peresini, and D. Kostic, "What you need to know about sdn control and data planes," tech. rep., 2014.

[21] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *International Conference on Passive and Active Network Measurement*, pp. 347–359, Springer, 2015.

[22] K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Latency in software defined networks: Measurements and mitigation techniques," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 435–436, 2015.

[23] Z. Bozakov and A. Rizk, "Taming sdn controllers in heterogeneous hardware environments," in *2013 Second European Workshop on Software Defined Networks*, pp. 50–55, IEEE, 2013.

[24] R. Durner, A. Blenk, and W. Kellerer, "Performance study of dynamic qos management for openflow-enabled sdn switches," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pp. 177–182, IEEE, 2015.

[25] 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*, pp. 199–212, 2014.

[26] 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*, pp. 43–48, 2013.

[27] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "Oflops: An open framework for openflow switch evaluation," in *International Conference on Passive and Active Network Measurement*, pp. 85–95, Springer, 2012.

[28] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring control plane latency in sdn-enabled switches," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pp. 1–6, 2015.

[29] G. Pongrácz, L. Molnár, and Z. L. Kis, "Removing roadblocks from sdn: Openflow software switch performance on intel dpdk," in *2013 Second European Workshop on Software Defined Networks*, pp. 62–67, IEEE, 2013.

[30] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "Openflow switching: Data plane performance," in *2010 IEEE International Conference on Communications*, pp. 1–5, IEEE, 2010.

[31] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pp. 120–125, IEEE, 2014.

[32] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *2011 23rd International Teletraffic Congress (ITC)*, pp. 1–7, IEEE, 2011.

[33] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an openflow switch on the netfpga platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 1–9, 2008.

[34] A. Gelberger, N. Yemini, and R. Giladi, "Performance analysis of software-defined networking (sdn)," in *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 389–393, IEEE, 2013.

[35] Y.-D. Lin, Y.-K. Lai, C.-Y. Wang, and Y.-C. Lai, "Ofbench: Performance test suite on openflow switches," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2949–2959, 2017.