What you need to know about (Smart) Network Interface Cards

Georgios P. Katsikas[1] [0000−0002−3890−6583], Tom Barbette[1] [0000−0003−1269−2190], Marco Chiesa[1] [0000−0002−9675−9729], Dejan Kostić[1] [0000−0002−1256−1070], and Gerald Q. Maguire Jr.[1] [0000−0002−6066−746X]

KTH Royal Institute of Technology, Sweden
{katsikas,barbette,mchiesa,dmk,maguire}@kth.se

Abstract. Network interface cards (NICs) are fundamental components of modern high-speed networked systems, supporting multi-100 Gbps speeds and increasing programmability. Offloading computation from a server's CPU to a NIC frees a substantial amount of the server's CPU resources, making NICs key to offer competitive cloud services. Therefore, understanding the performance benefits and limitations of offloading a networking application to a NIC is of paramount importance.
In this paper, we measure the performance of four different NICs from one of the *largest NIC vendors worldwide*, supporting 100 Gbps and 200 Gbps. We show that while today's NICs can easily support multi-hundred-gigabit throughputs, performing frequent update operations of a NIC's packet classifier — as network address translators (NATs) and load balancers would do for each incoming connection — results in a dramatic *throughput reduction of up to 70* Gbps *or complete denial of service*. Our conclusion is that all tested NICs cannot support *high-speed* networking applications that require keeping track of a large number of frequently arriving incoming connections. Furthermore, we show a variety of counter-intuitive performance artefacts including the performance impact of using multiple tables to classify flows of packets.

Keywords: Network interface cards · hardware classifier · offloading · rule operations · performance · benchmarking · 100 GbE.

1 Introduction

With the dramatic growth of Network Interface Card (NIC) speeds, optimizing I/O operations is essential for supporting modern-day applications. As evidenced by recent work, handling 40 Gbps of Transmission Control Protocol (TCP) traffic requires roughly 20%-60% of the CPU resources on a general-purpose server [10,31,48]. These communication overheads consume CPU cycles that could otherwise be used to run customers' applications, ultimately resulting in expensive deployments for network operators.

This is the preprint version of our paper presented at PAM'21. The final authenticated version is available online at https://doi.org/10.1007/978-3-030-72582-2_19

Offloading network operations to NICs is a pragmatic way to *partially* relieve CPUs from the burden of managing (some of the) network-related state. Examples of such offloading are TCP optimizations, such as Large Receive Offload (LRO) and TCP Segmentation Offload (TSO) [1]. Increasingly, NICs are equipped with built-in Field-Programmable Gate Arrays (FPGAs) or network processor cores that can be used to offload computation from a host's CPU directly into the NICs. Such NICs are referred to as SmartNICs. Several preliminary investigations of SmartNIC technologies have demonstrated potential benefits for offloading networking stacks [2,10,30,31,32], network functions [3,18,25,43,4], key-value stores[7,26,28], packet schedulers [44], neural networks [42], and beyond [21,38]. Despite the increasing relevance of (smart) NICs in today's systems, very few studies have focused on dissecting the performance of SmartNICs, comparing them with their predecessors, and providing guidelines for deploying NIC-offloaded applications, with a focus on packet classification.

Our goal. In this work, we study the performance of (smart) NICs for widely deployed packet classification operations. A key challenge of *packet classification* is the ability of the classifier to both quickly (*i*) match incoming packets to their packet processing actions and (*ii*) adapt the state of the packet classifier, e.g., by inserting new rules or updating existing ones. For example, consider a cloud load balancer (LB) that keeps track of the mapping between incoming connections and the back-end servers handling these connections. The LB may utilize a NIC's packet classifier to map TCP/IP 5-tuples of incoming connection identifiers to their corresponding servers. As a single cluster in a large-scale datacenter may receive over 10 million new connections per second [29], it is critical to support fast updates for packet classifiers, thus achieving high throughput and low predictable latency. Our study of packet classifiers reveals unexpected performance bottlenecks in today's (smart) NICs and provides guidelines for researchers and practitioners, who wish to offload dynamic packet classifiers to (smart) NICs.

Findings. We analyzed the performance of four different NICs with speeds in the 100 Gbps to 200 Gbps range. Our key findings are summarized in Table 1. In short, we show that the forwarding throughput of the tested NICs sharply degrades when *i)* the forwarding plane is updated and *ii)* packets match multiple forwarding tables in the NIC. Moreover, we devise an efficient in-memory update mechanism that mitigates the impact of updating the rules on the forwarding throughput. The code to reproduce the experiments of this paper is publicly available along with supplementary graphs showing the experimental evaluation of all four NICs under test [17].

Paper outline. This paper is organized as follows: §2 outlines the experimental methodology used in this work; §3 provides useful performance insights into modern NICs; §4 discusses related efforts in the area of programmable networking hardware beyond the work mentioned inline throughout the paper. Finally, §5 concludes this paper.

Table 1: Main findings of this paper.

Finding	**Implication**
There are parts of the NIC table hierarchy that do not yield the expected forwarding performance (§3.1).	Throughput degradation from 100 Gbps to 20 Mbps and multi-fold latency increase (Fig. 2a and Fig. 2c).
Uniformly spreading rules across a chain of NIC tables incurs performance penalty (§3.1).	Throughput degradation from 100 Gbps to 13 Gbps and 10x higher latency when using 16 tables (Fig. 2b and Fig. 2d).
A batch update of the NIC classifier, while processing traffic, makes the NIC temporarily unavailable (§3.1).	100% packet loss for up to several seconds with an increasing number of installed rules (Fig. 3).
Frequent updates of the NIC classifier, while processing traffic, causes substantial performance degradation (§3.1).	Throughput degradation from 100 Gbps to 30 Gbps and ~2x higher latency (Fig. 4).
Updating the NIC classifier from a separate core does not degrade the NIC performance (§3).	No performance impact when processing traffic on core 0 and updating rules from core 1 (Fig. 3 and Fig. 4).
The Internet protocol selection (i.e., IPv4 vs. IPv6) affects the NIC rule installation rate (§3.2.1).	IPv6 rule insertion rate is either 5-181x faster or 12% slower than the respective IPv4 rate, depending on the part of the NIC table hierarchy applied (Fig. 5a-5b).
The network slicing protocol selection affects the NIC rule installation rate (§3.2.1).	Installing VLAN-based rules is up to 50% faster than installing tunnel-based rules (Fig. 5c).
NIC rule update operations are non-atomic and rely on sequential addition and deletion (§3.2.2).	Too slow for applications that require heavy updates. Our dedicated update API performs up to 80% faster (Fig. 6).

2 Measurement Methodology

This section outlines the testbed used to conduct the experiments as well as our methodology to extract results.

2.1 Experimental Setup

Testbed. All of the experiments described in this paper used the testbed shown in Fig. 1. Two back-to-back interconnected servers, each with a dual-socket 16-core Intel®Xeon® Gold 6134 (SkyLake) CPU clocked at 3.2 GHz and 256 GiB of DDR4 Random Access Memory (RAM) clocked at 2666 MHz. Each core has 2×32 KiB L1 (instruction and data caches) and a 1 MiB L2 cache, while one 25 MiB Last Level Cache (LLC) is shared among the cores in each socket. Following today's best practices, hyper-threading is disabled on all servers [47] and the Operating System (OS) is the Ubuntu 18.04.5 distribution with Linux kernel v4.15. One server acts as a traffic generator and receiver while the other server is the Device Under Test (DUT).

Tested NICs. We focus our study on one of the most widespread NICs available in Commercial off-the-shelf (COTS) hardware to date, as shown in Table 2. Such

NICs, manufactured by NVIDIA Mellanox, operate at 100 Gbps link speeds (or beyond), while providing advanced processing capabilities. We also considered existing Intel NICs, such as the 10 GbE 82599 [12] and the 40 GbE XL710 [13], however these NICs operate at much lower link speeds and are limited to 8 K flow rules. The upcoming 100 GbE Intel E810 series network adapter [14] provides 16 K (masked) filters based on ternary content addressable memory (TCAM), which is still far from the range of several millions of flow rules tested with the NVIDIA Mellanox NICs. Moreover, the hardware limits of the Intel NICs are known, as Intel published relevant hardware datasheets [12,13,14]. NVIDIA Mellanox has not disclosed such information; thus our study sheds some light on unknown aspects of these popular NICs, while helping to understand how performance has evolved across the same family of NICs.

Table 2: The characteristics of the NICs used for the experiments in this paper.

Vendor	Model	Speed (Gbps)	# of Ports	Firmware Version	Driver	
					Name	Version
NVIDIA Mellanox	ConnectX-4 [35]	100	2	12.28.2006	mlx5_core	5.2-1.0.4
	ConnectX-5 [36]			16.29.1016		
	BlueField [34]			18.29.1016		
	ConnectX-6 [37]	200		20.29.1016		

All NICs except for the NVIDIA Mellanox ConnectX-6 use a PCIe 3.0 x16 bus to connect with a server's CPU. The ConnectX-6 adapter uses two PCIe 3.0 x16 slots. The BlueField NIC is a SmartNIC based on the ConnectX-5 adapter, also equipped with a 16-core ARM processor for additional in-NIC traffic processing. We briefly describe the general architecture and differences of the NVIDIA Mellanox NICs. All NICs have a first table, called *Table 0* or "root" table with space for 65 536 rule entries. All the NICs, except for the ConnectX-4, provide an additional sequence of high-performance exact-match tables (supporting a per-table mask) that can be used to massively offload packet classification from the CPUs to the NIC. Note that these NICs do not support Longest Prefix Match (LPM); instead the user should implement LPM with a combination of multiple tables with different masks. The capacity of these tables is only bounded by the host's available memory, thus they can accommodate a much larger number of rules, given the ample amount of RAM in modern servers. We refer to the first of those extra tables as *Table 1* and note that any subsequent table (i.e., Table 2,3, etc.) appears to have similar properties with Table 1.

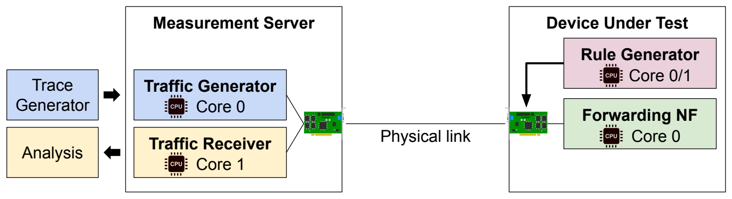


Fig. 1: Testbed setup and measurement methodology.

Traffic characteristics. A multi-core traffic generator and receiver, based on the Data Plane Development Kit (DPDK) v20.11 [46], is deployed on the measurement server as shown in Fig. 1. Four cores are allocated to the traffic generator, which inject a trace of 10K UDP flows at 100 Gbps. Each flow consists of MTU-sized (i.e., 1500-bytes) packets. This traffic first traverses the DUT and, if not dropped, then returns to the measurement server, this time reaching four different cores on the same CPU socket of the traffic generator.

Note that the measurement server injects traffic towards the DUT using the same 100 GbE ConnectX-5 NIC for all the experiments. This ensures that only the DUT's NIC hardware may vary across all of the experiments , thus potential differences among the experimental results solely depend on the performance of the underlying NIC in the DUT.

Measurements. Each experiment is executed 5 times; the collected measurements are plotted using either errorbars or boxplots, which visualize the 1st, 25th, 50th (i.e., median), 75th, and 99th values obtained across these 5 iterations, unless stated otherwise. The traffic receiver of the measurement server reports measurements related to end-to-end throughput, latency variance percentiles, per-queue packet & byte counters both at the measurement server and the DUT, packet loss, and the duration of each experiment. When reporting latency, we repeated experiments at 5Mpps (∼60 Gbps), avoiding link speed to be a bottleneck on both the DUT and the traffic generator, thus ensuring latency changes are due to the NIC and not packets buffering in queues.

3 Analysis of Flow Tables

This section benchmarks the selected NICs focusing on three different aspects related to packet classifiers.

First, we quantify the performance impact of the NICs' hardware classifiers with (i) an increasing number of rules, (ii) an increasing number of tables hosting these rules, and (iii) increasingly larger or more frequent updates being installed by the control plane(see §3.1). Second, we analyse the performance of flow rule insertion/deletion operations in terms of latency for rule insertions and throughput (see §3.2.1). Finally, after discovering flow rule modifications are not supported by these NICs, we evaluate a different strategy to realize fast and atomic rule updates in the packet classifier of the analyzed NICs (see §3.2.2).

3.1 Hardware Classification Performance

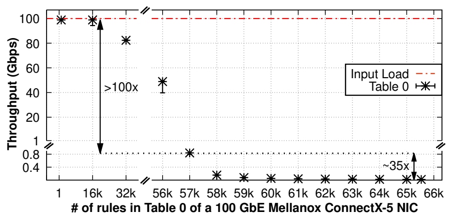
Overview. In this section we measure packet classification performance of modern NICs under a variety of conditions. First, we show that the first table of these NICs drops almost all traffic when memory utilization exceeds ∼85%. We also show that the packet processing latency of the analyzed NICs exhibits a long tail in this situation (up to 120 ms). Moreover, spreading an increasing number of rules across four or more tables in these NICs results in substantial throughput degradation (23-88% when using 4-16 tables). Finally, we show that runtime modifications to the packet classifier's rules have a detrimental effect on the NIC's throughput: we observe a reduction of 70 Gbps of throughput (out of 100 Gbps).

Scenario. In the following experiments the DUT runs a single-core forwarding Network Function (NF) using the testbed described in §2.1. The NIC of the DUT dispatches input frames to this NF according to the flow rules installed in the NIC. These rules are stored either in the default "root" flow table of the NIC (i.e., Table 0) or in non-root tables (i.e., Tables 1-16). We differentiate between these two table categories as NVIDIA Mellanox explicitly mentions that Table 0 has a limited number of supported flow entries (i.e., 2^{16} rules) and the latter support a faster API based on shared memory between the NIC and the driver running in userlevel. We only show results for the ConnectX-5 NIC as we observe qualitatively similar trends for all the other NICs.

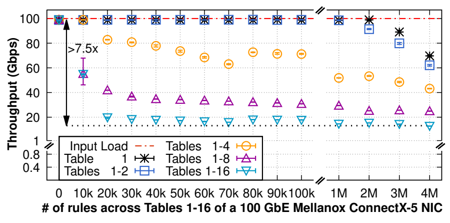
The rest of this section provides experimental evidence to address the following questions:

Q1 Does the number of rules and/or tables affect the performance of the NIC?

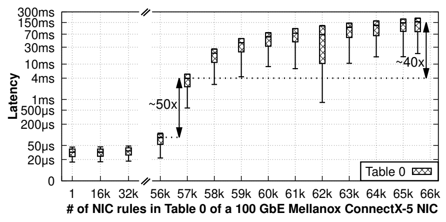
Figure 2 shows the performance of the packet classifier with an increasing number of rules (x-axis) for all types of tables of the NVIDIA Mellanox ConnectX-5 NIC. We denote by Table 1-X the case where we uniformly install forwarding rules on the first X non-root tables, i.e., Table 1, …, Table X. The rules installed in the NIC are simple exact matches and the generated traffic matches exactly one



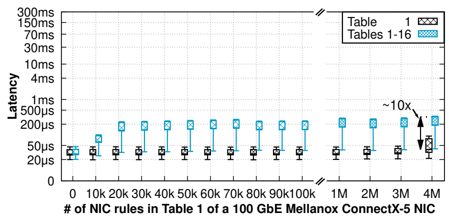
(a) Throughput (Table 0).



(b) Throughput (Tables 1 to 16).



(c) Latency (Table-0).



(d) Latency (Tables 1 to 16).

Fig. 2: Throughput and latency (on a logarithmic scale) of a hardware-based 100 GbE NVIDIA Mellanox ConnectX-5 NIC classifier with different number of pre-installed rules across Table 0 (left) and Tables 1-16 (right).

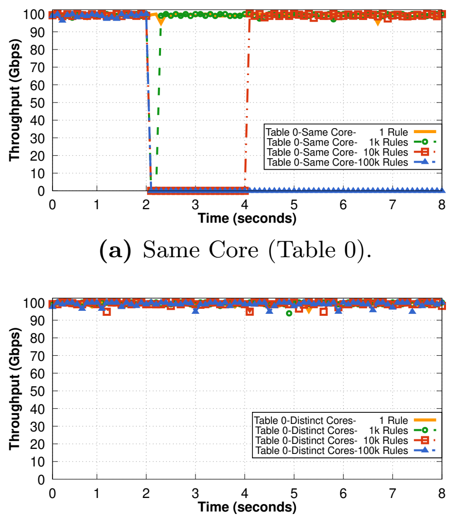
default rule installed in the NIC. We generate 8Mpps of 1.5KB packets towards the DUT, equivalent to 100 Gbps. Fig. 2a and 2c show that the performance (i.e., throughput and packet processing latency) for Table 0 decreases dramatically as soon as the occupancy of the table goes above 85%, hence the last 15% of memory is in practice unusable. Specifically, the throughput decreases from 100 Gbps down to 20 Mbps, while the latency increases by several orders of magnitude, from tens of µs to more than a hundred of ms. We observe a similar decrease in throughput for small packets (i.e., 64B), even when the input load is 3.5 Gbps, which is 30x lower than the maximum attainable throughput of the NIC under test. This confirms that the performance degradation issue is not a result of excessive input load, but rather a design artifact of the root table.

Figures 2b and 2d show that non-root tables (i.e., Tables 1-16) are much faster than the root table. Specifically, using a non-root table the NIC achieves line-rate throughput and low predictable latency even with $2M$ entries in Table 1. However, spreading rules across an increasing number of non-root tables results in substantial performance degradation. As shown in Fig. 2b, for most of the tested ruleset sizes, the NIC cannot achieve more than 20 Gbps throughput when using 16 tables, while the respective latency to access these tables exhibits a ten-fold increase compared to the single-table case, as shown in Fig. 2d.

Q2 Do updates to the classifier affect the performance of the NIC?

The objective of this experiment is to understand how runtime modifications of the packet classifier's ruleset impact the throughput of the forwarded traffic. We envision two types of experiments motivated by two different use cases. In the first experiment, we generate a single *batch* of rule insertions to be installed into the NIC. This is reminiscent of scenarios in which a network suddenly reacts to a failure event that triggers many rule updates. For instance, Internet link failures may generate a burst of BGP updates for possibly 10s of 100s of thousands of IP prefixes received from a neighboring network [11]. In the second experiment, we generate periodic rule insertions in the packet classifier at a given frequency. This setting is reminiscent of cloud datacenter Layer 4 load balancers (LBs), where LBs insert a new rule into a packet classifier each time a new connection arrives. We note that, based on realistic connection size distributions taken from cloud datacenter workloads, the number of new rules to be installed ranges between 4K per second for "Hadoop' workloads to 36K and 338K per second for "cache follower" and "web server" workloads, respectively [41]. In both experiments, we generate a workload with packet sizes of 1.5 KB. To avoid external bias from the system's CPU, we measure two different cases for each experiment: In the first case (labeled as "Same Core" below), we use the same CPU core that performs traffic forwarding to install the rules in the NIC. In the second case (labeled as "Distinct Cores" below), we use one CPU core for traffic forwarding and another CPU core for rule installation. All the traffic matches a single rule in the classifier. As in the previous experiment, we obtain similar qualitative results for all the NICs and only show the NVIDIA ConnectX-5 ones.

Batch-based updates have detrimental effects on performance. Figure 3 shows the packet processing throughput (y-axis) achieved by the NIC's packet classifier over time (x-axis) for Tables 0 and 1, while the NIC simultaneously (*i*) receives a workload of 100 Gbps of 1500 B packets and (*ii*) inserts a number of new rules (see the legends) ranging between 1 and 100 K.



(a) Same Core (Table 0). **(b)** Same Core (Table 1).

(c) Distinct Cores (Table 0). **(d)** Distinct Cores (Table 1).

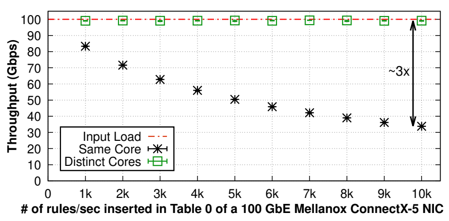
Fig. 3: Impact of batch-based updates on the performance of a 100 GbE NVIDIA Mellanox ConnectX-5 NIC classifier.

As shown in Fig. 3a even with a batch of 1K rules (see the green circles), the NIC fails to process any traffic for about 300 ms. For a 100 Gbps link with MTU-sized frames, this translates to a packet loss of around 2.5 M frames, while more than 40 M frames could have been lost from a 100 Gbps link with 64 B frames. Increasing the rules' batch size to 10K results in a longer failure of around 2 s (see the red squares), while in the case of 100K rules (blue triangles) the NIC does not recover even after 6 s. The down-time of Table 1 is 500 ms, but the problem manifests itself only in the case of 100 K rules as shown in Fig. 3b. On the other hand, installing the batch updates from a dedicated core does not affect the forwarding performance of the NF as shown in Fig. 3c and 3d.

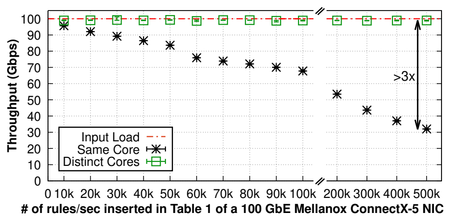
We believe that these results have far-reaching implications on both (*i*) the security of the network functions, as batch-based updates could become a vector of denial-of-service attacks and (*ii*) the design of highly-reactive network controllers, e.g., to enable large data-plane updates for fast failover recovery [6].

Rate-based updates reduce NIC forwarding capacities. Installing periodic batches of rules from the same core is a typical operation of NATs and Layer 4 load balancers, which need to reactively install rules matching new incoming connections. Installing rules from a different core allows us to dissect just the performance degradation due to interference in the NIC data-plane.

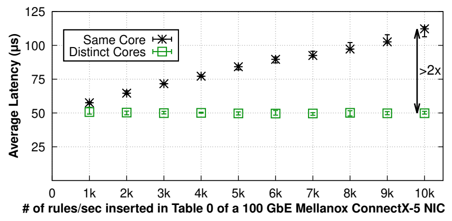
Fig. 4a and 4b show the throughput of the forwarding NF when we simultaneously insert rules into the NIC classifier at a specific rate. The insertion rate ranges between 1 K to 10 K rules per second for Table 0 and 10 K to 500 K rules per second for Table 1. The inserted rules are not generated in response to a



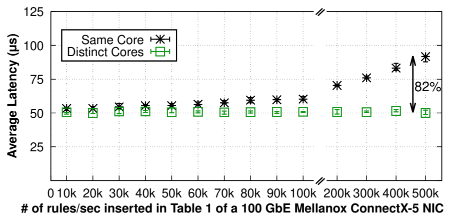
(a) Throughput (Table 0).



(b) Throughput (Table 1).



(c) Average Latency (Table 0).



(d) Average Latency (Table 1).

Fig. 4: Impact of rate-based updates on the performance of a 100 GbE NVIDIA Mellanox ConnectX-5 NIC classifier.

new incoming connection but pre-computed and inserted regardless of when new connections arrive. The results show that when inserting rules from a different core, the throughput and average latencies (see also Fig. 4c and 4d) are mostly unaffected by the parallel insertion. However, when the insertions are generated from the same core running the forwarding NF, we observe a significant performance drop. Specifically, Fig. 4a and 4b show that the throughput decreases by roughly 70 Gbps for 10K and 500K rule insertions per second in Table 0 and Table 1, respectively. As shown in Fig. 4c and 4d, the respective latency increase is up to more than 2x for Table 0 and 82% for Table 1. This result demonstrates that the bottleneck of the update operation is the standard API provided by the NIC vendor for updating the forwarding table (which requires long time and interrupts the normal forwarding for prolonged period of times).

We note that installing rules from a different core is not a panacea. One would need expensive inter-core communication to install a rule as well as reserve extra CPU resources to handle the rule installation. For instance, to install i.e., 500K rules consumes 100% of a CPU core for several hundreds of milliseconds.

Summary. Our results show that it is possible to introduce a denial of service attack to the packet classifier of the NICs under test or dramatically reduce its throughput *by up to* 70 Gbps, by updating the classifier's rules using the same CPU core that performs traffic processing. This technique is commonly used by sharded high-speed data planes [4], as it would be the case for per-connection NFs. We note that realistic datacenter workloads generate new connections in the range of 4K-400K new connections per second. Our results indicate that one would *not* gain any benefit from offloading applications, such as cloud NATs and load balancers, with highly dynamic tables, to the analyzed NICs. Moreover, all NICs under test achieve similar performance across all the experiments in this section; with the only difference being the NVIDIA ConnectX-6 NIC, which exhibits slightly lower throughput degradation than the rest of the NICs in the experiment shown in Fig. 2a and 2b [17].

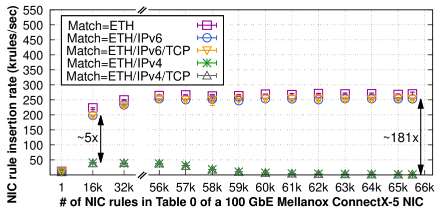
In the next subsection, we investigate how rule modifications are performed and explore performance limitations of these rule modifications. In response to this, we provide alternative workarounds that mitigate some of the issues described in this section.

3.2 Rule Operations Analysis

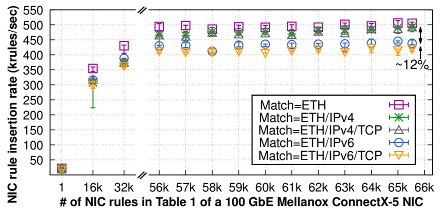
We now focus solely on the performance of rule update operations (i.e., insertions, deletions, and modifications' completion times). Clearly, the shorter the update completion time, the lower the performance disruption on the forwarded traffic. Our analysis reveals three main findings. First, while modern NICs handle almost 500K insertions per second, there is a significant and sometimes counter-intuitive performance difference depending on the type and number of fields that are matched by the packet classifier, as well as the type and number of actions that are applied by a rule. Surprisingly, installing rules matching IPv4 in Table 0 is a much slower process than installing IPv6 rules. Our second finding is that the cost of installing VLAN-based rules for network slicing is substantially lower than the respective cost of installing GRE/VXLAN/GENEVE-based rules. Our third and final finding relates to the fact that rule modification operations are not atomically supported by the analyzed NICs: one has to delete the old rule and insert a new one. Our analysis shows that rule modification time can be decreased by 80% compared to the insertion/deletion operations supported by the standard API of the vendors, by directly modifying the content of the exact match tables in the NIC memory.

3.2.1 Insertion/deletion of rules We now compute the rule insertion rate supported by an NVIDIA Mellanox ConnectX-5 NIC in Tables 0 and 1. We use a single CPU core to insert a number of rules in the range between 1 and 65536 and measure the time that it takes to insert them. From this value, we compute the rule insertion rate.

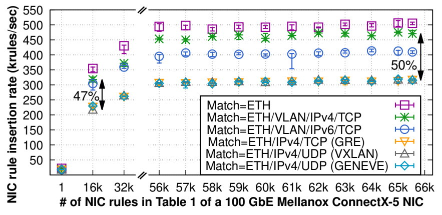
Figure 5a shows that the rules insertion rate for Table 0 for five types of rules matching different combinations of fields, such as Ethernet, IPv4, IPv6,



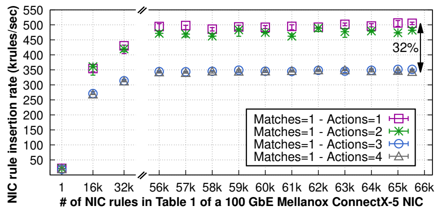
(a) Rule insertion rate (Table 0) between IPv4 and IPv6 rules.



(b) Rule insertion rate (Table 1) between IPv4 and IPv6 rules.



(c) Rule insertion rate (Table 1) for various slicing protocols.



(d) Rule insertion rate (Table 1) with increasing number of actions.

Fig. 5: Rule insertion performance (in kRules/sec) of a hardware-based 100 GbE NVIDIA Mellanox ConnectX-5 NIC classifier with various rule sets in [1, 65536] stored in two different tables.

and TCP. A single action is applied to a packet matching any of these rules. Surprisingly, our measurements show a striking difference between IPv4 and IPv6. Specifically, inserting rules matching IPv4 results in a sharp slow-down in the insertion rate compared to IPv6 rules, which is already 5x slower with just $16K$ entries. We profiled both operations to unveil the reasons of this performance diversity and found that IPv4 rules are directly installed by the kernel in hardware, using the firmware API, while the IPv6 rules are managed by the userlevel DPDK driver similarly to the rules of non-root tables. On the contrary, Fig. 5b shows the same experiment for Table 1. We note that in this case matching on IPv4 results in a 12% higher insertion rate compared to IPv6, the opposite of what we observed for Table 0. This is because Table 1 is managed in software, thus both IPv4 and IPv6 are managed by the respective DPDK driver.

We then investigate how different extensively adopted network slicing protocols affect the insertion rate into the NIC's most performant Table 1. We consider VLAN, GRE, VXLAN, and GENEVE virtualization headers, which are widely used in datacenter and wide-area network deployments. Figure 5c shows that rules matching VLAN tags can be installed up to 50% more rapidly than those relying on the other virtualization schemes.

We now verify whether the extent to which the actions associated to the rules impact the rules' insertion rate. Figure 5d shows that increasing the number of actions performed on a packet may result in 32% slower insertion rate. We believe these results are inline with the natural intuition of slower insertions for more complex actions.

We finally repeat all the previous experiments but in this case we remove entries from the NIC's packet classifier. To our biggest surprise, when we add a TCP match on a set of rules in Table 0, the deletion becomes faster than without having the TCP header. This counter-intuitive result demonstrates once more that any deployment on Table 0 should be accompanied by a comprehensive testing of the classifier's structure to avoid unexpected performance slowdown.

3.2.2 Modification of rules We now investigate the problem of *updating* a set of rules on the analyzed NICs. We first observe that `none` of the evaluated NICs support direct flow modifications through their APIs. One has to first delete and then insert an entry, which results in two major issues: (*i*) there are periods when the network configuration is incorrect and (*ii*) as observed in the previous subsection, rule modifications are extremely slow for the needs of real platforms. We therefore show how one can carefully engineer flow modifications for simple 5-tuple matching rules to speed up rule modifications in the NIC. We refer to our technique as *enhanced in-memory update*. Our technique does not rely on the standard API provided by the NIC vendor in DPDK and the rdma-core library to modify rules, but instead directly accesses the memory of the exact-match stages in the pipeline and modifies them in a less disruptive way. We defer the reader later in this section for more details on our improved update technique.

Enhanced in-memory updates are up to 80% faster. We employed DPDK's *flow-perf* tool to measure the NICs update rate, using the standard sequential deletion and insertion process. Then, we modified this tool to update all installed rules by using our in-memory update. Figure 6 shows the update rate (y-axis) in krules/sec achieved by both (i) the standard API deletion/insertion (black squares) and (ii) our enhanced in-memory update scheme (blue stars) with an increasing number of rules (x-axis) in the NIC classifier.

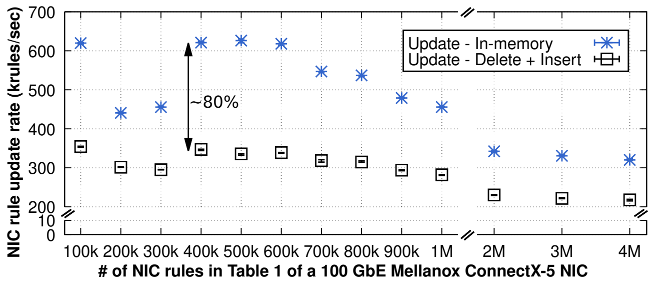


Fig. 6: Evaluation of the enhanced in-memory update mechanism.

We note that the standard API achieves 300K TCP/IP flow updates per second[*] on average, possibly disrupting all the forwarded traffic as shown in §3.1 in Q2. Our enhanced in-memory updates of the NIC classifier increases the insertion rate for TCP/IP rules by up to 80%. We observe the CPU stalls during the experiment, waiting for the NIC to complete memory synchronization commands, hence reaching the limit of the NIC Direct Memory Access (DMA) engine. We leave the problem of making the update mechanism more generic, possibly in collaboration with NVIDIA Mellanox, as future work.

Enhanced in-memory updates explained. We now explain more in detail how one can realize faster rule insertions/deletions/modifications on the analyzed NICs. While NIC vendors provide their own standard API for rule modifications, we added our own API for rule updates in DPDK and implemented support for the API in the `mlx5` driver (supporting ConnectX-4 and higher NVIDIA Mellanox NICs), and the backing `rdma-core` library that handles messaging between the NIC driver and the NIC itself. Instead of inserting and then removing a rule, our new API is based on efficient in-memory updates to avoid as many memory allocations in the driver as possible, while reusing the data-structure and only changing the match/action field values. In the ConnectX-5 and above NICs, a rule in a non-root table is implemented using a series of exact match hash-tables. Each hash-table only supports a unique mask, i.e., it is left to the user to implement techniques, such as tuple-space search, to implement an efficient LPM strategy by using a series of various exact-match masked prefixes.

In the case of standard TCP 5-tuple, one needs *two* hash tables. The first table matches the IP version, and the second one matches the 5-tuple itself. As far as our reverse-engineering of this undocumented mechanism can tell, this separation of the header fields into multiple hash-tables is dictated by the firmware, which supports a certain set of groups of fields per hash-table. Each of these two hash-tables work on one of those group of fields, eventually masked. Adding more fields from the application layer, or diverse tunnel types (VXLAN, GRE, etc.) will add more hash-tables in the chain. We note that the NIC handles hash-table collisions using a per-bucket linked-list of colliding entries. When inserting a new TCP/IP 5-tuple, the standard API would take an atomic reference on the entry for the IP-version and insert an entry into the 5-tuple hash-tables, and then remove the old rule from both hash-tables. Our update mechanism tries to minimize the number of modifications by following the existing rule, leaving it in the same place when the bucket does not change, not changing atomic reference (as it is the case for the IP-version hash-table), and then we either rewrite in-place the bucket of the hash-table if the bucket index (i.e., a CRC32 hash of the masked fields value) did not change (it is probable as all hash-tables start with a very small size and grow as needed), or move the entry in the pointed bucket to a new bucket if the index changes. This also avoids multiple calls to the DMA engine to insert and remove the rule, by only selectively updating the memory zone of the field that changed, as well as limiting the number of

[*]The employed DPDK v20.11 flow API is single-threaded. Higher performance could be achieved using multi-threaded rule insertion/deletion added in DPDK v21.02 [15].

memory accesses. As far as consistency matters, our approach tries to guarantee atomicity of the update in the NIC. There exists a small amount of time during which, when the bucket entry is moved, the old and the new entries co-exist before the old entry is marked as invalid. We believe this co-existence does not open a security vulnerability since both entries are valid. Operators should fall back to the standard API if this is a concern.

For now, we only support updates on match operations' values; to implement action operations' value updates, such as redirecting packets to a different queue, would be fairly similar. This is particularly suitable for connection tracking, such as NAT and load balancers. Updating the masks of a rule is another complex operation that we currently do not support. This is challenging because different hash-tables implement different masked elements, possibly defeating the benefits of re-using some pre-installed elements.

4 Related Work

Measuring the performance of emerging network technologies has brought enormous benefits in our understanding of where the critical bottlenecks reside in today's deployed network systems. Neugebauer et al [33] have investigated the performance of the PCIe device interconnecting modern NICs to the CPUs and memories showing surprisingly low performance with small packets. Farshin et al. (*i*) quantified the impact of direct cache access in Intel processors [9] and (*ii*) proposed software stack optimizations to achieve per-core hundred-gigabit networking [8]. Kuzniar et al [23,24] have unveiled a variety of issues with the initial OpenFlow-based switches, such as the lack of consistency during updates. In contrast, we focus on NIC performance. Liu et al. [28] analyzed the memory characteristics, number of cores required to forward a certain amount of traffic, and Remote Direct Memory Access (RDMA) capabilities of five different SmartNICs. In our work, we focus on the packet classifier component of a NIC and the impact of memory occupancy and runtime modifications on its performance.

A variety of efforts have been devoted to the orthogonal problem of scheduling updates in a network [16,19,20,22,27,40,39] or designing faster data structures at the data-plane level that are amenable to quick modifications [5,6,45].

5 Conclusions

Motivated by the ever-growing increase of networking speeds and offloading trends, this paper investigates the performance bottlenecks of today's NIC packet classifiers. We focused on several evolving models of one of the largest NIC vendors worldwide, showing a variety of critical performance limitations depending of the memory occupancy, the pipeline length, runtime rule modifications, and rule modification speed. We explored the idea of performing gradual updates directly in the NIC, improving the unveiled bottlenecks as well as many obstacles towards building a more efficient and generic API.

6 Acknowledgments

We would like to thank our shepherd Dr. Diego Perino and the anonymous reviewers for their insightful comments on this paper. This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 770889). This work was also funded by the Swedish Foundation for Strategic Research (SSF) and KTH Digital Futures.

References

1. Antichi, G., Callegari, C., Giordano, S.: Implementation of TCP large receive offload on open hardware platform. In: Proceedings of the 2013 ACM Workshop on High Performance and Programmable Networking. pp. 15–22 (2013). https://doi.org/10.1145/2465839.2465842
2. Arashloo, M.T., Lavrov, A., Ghobadi, M., Rexford, J., Walker, D., Wentzlaff, D.: Enabling Programmable Transport Protocols in High-Speed NICs. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). pp. 93–109. USENIX Association, Santa Clara, CA (Feb 2020), https://www.usenix.org/conference/nsdi20/presentation/arashloo
3. Ballani, H., Costa, P., Gkantsidis, C., Grosvenor, M.P., Karagiannis, T., Koromilas, L., O'Shea, G.: Enabling End-Host Network Functions. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. p. 493–507. SIGCOMM '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2785956.2787493
4. Barbette, T., Katsikas, G.P., Maguire, Jr., G.Q., Kostić, D.: RSS++: load and state-aware receive side scaling. In: Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies. pp. 318–333. CoNEXT '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3359989.3365412
5. Bonaventure, O., Filsfils, C., Francois, P.: Achieving sub-50 milliseconds recovery upon bgp peering link failures. IEEE/ACM Trans. Netw. **15**(5), 1123–1135 (Oct 2007). https://doi.org/10.1109/TNET.2007.906045
6. Chiesa, M., Sedar, R., Antichi, G., Borokhovich, M., Kamisiński, A., Nikolaidis, G., Schmid, S.: PURR: A Primitive for Reconfigurable Fast Reroute: Hope for the Best and Program for the Worst. In: Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies. p. 1–14. CoNEXT '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3359989.3365410
7. Eran, H., Zeno, L., Tork, M., Malka, G., Silberstein, M.: NICA: An Infrastructure for Inline Acceleration of Network Applications. In: Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference. p. 345–361. USENIX ATC '19, USENIX Association, USA (2019), https://www.usenix.org/system/files/atc19-eran.pdf
8. Farshin, A., Barbette, T., Roozbeh, A., Maguire Jr., G.Q., Kostić, D.: PacketMill: Toward per-core 100-Gbps Networking. In: Proceedings of the Twenty-Sixth International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS '21, Association for Computing Machinery, New York, NY, USA (2021), https://doi.org/10.1145/3445814.3446724

9. Farshin, A., Roozbeh, A., Maguire Jr., G.Q., Kostić, D.: Reexamining Direct Cache Access to Optimize I/O Intensive Applications for Multi-hundred-gigabit Networks. In: 2020 USENIX Annual Technical Conference (USENIX ATC 20). pp. 673–689. USENIX Association (Jul 2020), https://www.usenix.org/conference/atc20/presentation/farshin
10. Firestone, D., Putnam, A., Mundkur, S., Chiou, D., Dabagh, A., Andrewartha, M., Angepat, H., Bhanu, V., Caulfield, A., Chung, E., Chandrappa, H.K., Chaturmohta, S., Humphrey, M., Lavier, J., Lam, N., Liu, F., Ovtcharov, K., Padhye, J., Popuri, G., Raindel, S., Sapre, T., Shaw, M., Silva, G., Sivakumar, M., Srivastava, N., Verma, A., Zuhair, Q., Bansal, D., Burger, D., Vaid, K., Maltz, D.A., Greenberg, A.: Azure Accelerated Networking: SmartNICs in the Public Cloud. In: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). pp. 51–66. USENIX Association, Renton, WA (2018), https://www.usenix.org/system/files/conference/nsdi18/nsdi18-firestone.pdf
11. Holterbach, T., Vissicchio, S., Dainotti, A., Vanbever, L.: SWIFT: Predictive Fast Reroute. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. pp. 460–473 (2017). https://doi.org/10.1145/3098822.3098856
12. Intel: 82599 10 GbE Controller Datasheet (2016), http://www.intel.com/content/www/us/en/embedded/products/networking/82599-10-gbe-controller-datasheet.html
13. Intel: Ethernet Converged Network Adapter XL710 10/40 GbE (2016), https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-xl710-brief.pdf
14. Intel: Ethernet Controller E810 (2021), https://www.intel.com/content/www/us/en/design/products-and-solutions/networking-and-io/ethernet-controller-e810/technical-library.html
15. Jaddo Wisam: app/flow-perf: add multi threaded support (Jan 2021), https://inbox.dpdk.org/dev/20201126111543.16928-4-wisamm@nvidia.com/T/
16. Jin, X., Liu, H.H., Gandhi, R., Kandula, S., Mahajan, R., Zhang, M., Rexford, J., Wattenhofer, R.: Dynamic Scheduling of Network Updates. In: Proceedings of the 2014 ACM Conference on SIGCOMM. pp. 539–550. SIGCOMM '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2619239.2626307
17. Katsikas, G.P., Barbette, T.: GitHub repository hosting the NIC benchmarks and collected data, https://github.com/nic-bench
18. Katsikas, G.P., Barbette, T., Kostić, D., Steinert, R., Maguire Jr., G.Q.: Metron: NFV Service Chains at the True Speed of the Underlying Hardware. In: 15th USENIX Conference on Networked Systems Design and Implementation. pp. 171–186. NSDI'18, USENIX Association, Renton, WA (2018), https://www.usenix.org/system/files/conference/nsdi18/nsdi18-katsikas.pdf
19. Katta, N., Hira, M., Kim, C., Sivaraman, A., Rexford, J.: HULA: Scalable Load Balancing Using Programmable Data Planes. In: Proceedings of the Symposium on SDN Research. pp. 10:1–10:12. SOSR '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2890955.2890968
20. Katta, N.P., Rexford, J., Walker, D.: Incremental Consistent Updates. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. pp. 49–54. HotSDN '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2491185.2491191
21. Kim, D., Memaripour, A., Badam, A., Zhu, Y., Liu, H.H., Padhye, J., Raindel, S., Swanson, S., Sekar, V., Seshan, S.: Hyperloop: group-based NIC-offloading to accelerate replicated transactions in multi-tenant storage systems. In: Proceedings of

the 2018 Conference of the ACM Special Interest Group on Data Communication. pp. 297–312 (2018). https://doi.org/10.1145/3230543.3230572
22. Kuźniar, M., Perešíni, P., Kostić, D.: Providing Reliable FIB Update Acknowledgments in SDN. In: Proceedings of the 10th International Conference on Emerging Networking Experiments and Technologies. pp. 415–422. CoNEXT '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2674005.2675006
23. Kuźniar, M., Perešíni, P., Kostić, D.: What You Need to Know About SDN Flow Tables. In: Passive and Active Measurement (PAM). Lecture Notes in Computer Science, vol. 8995, pp. 347–359 (2015). https://doi.org/10.1007/978-3-319-15509-8_26
24. Kuźniar, M., Perešíni, P., Kostić, D., Canini, M.: Methodology, Measurement and Analysis of Flow Table Update Characteristics in Hardware OpenFlow Switches. Computer Networks: The International Journal of Computer and Telecommunications Networking, Elsevier, **vol. 26** (2018). https://doi.org/https://doi.org/10.1016/j.comnet.2018.02.014
25. Le, Y., Chang, H., Mukherjee, S., Wang, L., Akella, A., Swift, M.M., Lakshman, T.: Uno: uniflying host and smart NIC offload for flexible packet processing. In: Proceedings of the 2017 Symposium on Cloud Computing. pp. 506–519 (2017). https://doi.org/10.1145/3127479.3132252
26. Li, B., Ruan, Z., Xiao, W., Lu, Y., Xiong, Y., Putnam, A., Chen, E., Zhang, L.: Kv-direct: High-performance in-memory key-value store with programmable MIC. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 137–152 (2017). https://doi.org/10.1145/3132747.3132756
27. Liu, H.H., Wu, X., Zhang, M., Yuan, L., Wattenhofer, R., Maltz, D.: zUpdate: Updating Data Center Networks with Zero Loss. In: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM. pp. 411–422. SIGCOMM '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2486001.2486005
28. Liu, M., Cui, T., Schuh, H., Krishnamurthy, A., Peter, S., Gupta, K.: Offloading distributed applications onto smartNICs using iPipe. In: Proceedings of the ACM Special Interest Group on Data Communication, pp. 318–333. Association for Computing Machinery (2019). https://doi.org/10.1145/3341302.3342079
29. Miao, R., Zeng, H., Kim, C., Lee, J., Yu, M.: Silkroad: Making stateful layer-4 load balancing fast and cheap using switching ASICs. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. pp. 15–28 (2017). https://doi.org/10.1145/3098822.3098824
30. Mittal, R., Shpiner, A., Panda, A., Zahavi, E., Krishnamurthy, A., Ratnasamy, S., Shenker, S.: Revisiting network support for RDMA. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. pp. 313–326 (2018). https://doi.org/10.1145/3230543.3230557
31. Moon, Y., Lee, S., Jamshed, M.A., Park, K.: AccelTCP: Accelerating Network Applications with Stateful TCP Offloading. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). pp. 77–92. USENIX Association, Santa Clara, CA (Feb 2020), https://www.usenix.org/conference/nsdi20/presentation/moon
32. Narayan, A., Cangialosi, F., Raghavan, D., Goyal, P., Narayana, S., Mittal, R., Alizadeh, M., Balakrishnan, H.: Restructuring endpoint congestion control. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. pp. 30–43 (2018). https://doi.org/10.1145/3230543.3230553
33. Neugebauer, R., Antichi, G., Zazo, J.F., Audzevich, Y., López-Buedo, S., Moore, A.W.: Understanding PCIe performance for end host networking. In: Proceedings

of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018. pp. 327–341. ACM (2018). https://doi.org/10.1145/3230543.3230560

34. NVIDIA Mellanox: BlueField® SmartNIC for Ethernet (2019), https://www.mellanox.com/related-docs/prod_adapter_cards/PB_BlueField_Smart_NIC.pdf
35. NVIDIA Mellanox: ConnectX®-4 EN Card 100Gb/s Ethernet Adapter Card (2020), http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-4_EN_Card.pdf
36. NVIDIA Mellanox: ConnectX®-5 EN Card 100Gb/s Ethernet Adapter Card (2020), http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-5_EN_Card.pdf
37. NVIDIA Mellanox: ConnectX®-6 EN IC 200GbE Ethernet Adapter IC (2020), https://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX-6_EN_IC.pdf
38. Palkar, S., Abuzaid, F., Bailis, P., Zaharia, M.: Filter before you parse: Faster analytics on raw data with sparser. Proceedings of the VLDB Endowment **11**(11), 1576–1589 (2018). https://doi.org/10.14778/3236187.3236207
39. Perešíni, P., Kuźniar, M., Canini, M., Kostić, D.: ESPRES: Transparent SDN Update Scheduling. In: Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking. pp. 73–78. HotSDN '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2620728.2620747
40. Reitblatt, M., Foster, N., Rexford, J., Schlesinger, C., Walker, D.: Abstractions for Network Update. In: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. pp. 323–334. SIGCOMM '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2342356.2342427
41. Roy, A., Zeng, H., Bagga, J., Porter, G., Snoeren, A.C.: Inside the social network's (datacenter) network. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. pp. 123–137 (2015). https://doi.org/10.1145/2785956.2787472]
42. Siracusano, G., Galea, S., Sanvito, D., Malekzadeh, M., Haddadi, H., Antichi, G., Bifulco, R.: Running neural networks on the NIC (2020), https://arxiv.org/abs/2009.0235
43. Spaziani Brunella, M., Belocchi, G., Bonola, M., Pontarelli, S., Siracusano, G., Bianchi, G., Cammarano, A., Palumbo, A., Petrucci, L., Bifulco, R.: hXDP: Efficient Software Packet Processing on FPGA NICs. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, Banff, Alberta (Nov 2020), https://www.usenix.org/conference/osdi20/presentation/brunella
44. Stephens, B., Akella, A., Swift, M.: Loom: Flexible and Efficient NIC Packet Scheduling. In: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). pp. 33–46. USENIX Association, Boston, MA (Feb 2019), https://www.usenix.org/conference/nsdi19/presentation/stephens
45. Stephens, B., Cox, A.L., Rixner, S.: Scalable Multi-Failure Fast Failover via Forwarding Table Compression. In: Proceedings of the Symposium on SDN Research, SOSR 2016, Santa Clara, CA, USA, March 14 - 15, 2016. p. 9. ACM (2016), https://doi.org/10.1145/2890955.2890957
46. The Linux Foundation: Data Plane Development Kit (DPDK), http://dpdk.org, http://dpdk.org

47. Zhang, T., Linguaglossa, L., Gallo, M., Giaccone, P., Iannone, L., Roberts, J.: Comparing the performance of state-of-the-art software switches for NFV. In: Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies. pp. 68–81 (2019). https://doi.org/10.1145/3359989.3365415
48. Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., Padhye, J., Raindel, S., Yahia, M.H., Zhang, M.: Congestion control for large-scale RDMA deployments. ACM SIGCOMM Computer Communication Review **45**(4), 523–536 (2015). https://doi.org/10.1145/2785956.2787484