OFLOPS-SUME and the art of switch characterization

Rémi Oudin* Gianni Antichi§, Charalampos Rotsos‡, Andrew W. Moore†, Steve Uhlig§
*Université Paris-Saclay, †University of Cambridge, ‡Lancaster University, §Queen Mary, University of London

***Abstract*—The philosophy of SDN has introduced new challenges in network system management. In contrast to traditional network devices that contained both the control and the data plane functionality in a tightly coupled manner, SDN technologies separate the two network planes, and define a remote API for low-level device configuration. Nonetheless, the enhanced flexibility of the SDN paradigm is prone to create novel performance and scalability bottlenecks in the network.**

To help network managers and application developers better understand the actual behavior of SDN implementations, we present a hardware/software co-design that enables switch characterization at 40Gbps and beyond. We conduct an evaluation of both software and hardware switches. We expose the unwanted effects of the OpenFlow barrier primitive, potential misbehaviors when adding or modifying a batch of rules and how simple operations, such as packet modification, can impact the switch forwarding performance. We release the code publicly as open source to promote experiments reproducibility, as well as encourage the network community to evolve our solution.

***Index Terms*—OpenFlow, testing, switch performances**

I. Introduction

Research on Software-Defined Networking (SDN) technologies has produced a wide range of applications improving network functionality [1]. As a result, many network vendors, within only a few years, enabled SDN support in their products, in an effort to transfer research innovation into the market [2], [3].

However, SDN adoption has highlighted a series of new network management challenges. Legacy network devices contain both the control and the data plane functionality in a tightly coupled manner. In contrast, SDN technologies define a remote API for low-level device configuration, that clearly separates the control from the data plane. This API is used by a central network controller to implement new network services. The introduction of multiple abstraction layers in SDN network architectures introduces novel performance-critical functional blocks that can impact the overall network scalability. Specifically, network behavior can be affected by (1) the control stack, i.e., the control application and the Network Operating System (NOS), (2) the switch SDN driver, *i.e.* the interface between the control and data planes in an SDN switch, and (3) the switch silicon itself (Figure 1).

While much research has focused on control application and NOS benchmarking [4], [5], [6], [7], [8], we argue that the effective deployment of SDN in production networks requires a flexible and high–precision open–source switch performance characterization platform. The OpenFlow protocol, the predominant SDN realization, currently in its 1.6 version, has greatly morphed since its original versions. This evolution reflects the development and deployment experience of the technology, on a constantly widening range of network environments. Providing an open and flexible solution for SDN testing is important to ascertain that the behavior of specific implementations is in-line with the expectations of the supported OpenFlow versions. The SDN community requires a performance testing platform, capable to co-evolve with the protocol and to support rapid prototyping of experimentation scenarios that test and troubleshoot the impact of new protocol features. Furthermore, as network link speeds continuously increase, meaningful packet-level measurements have higher precision requirements, creating a challenge that can only be met with specialized hardware.

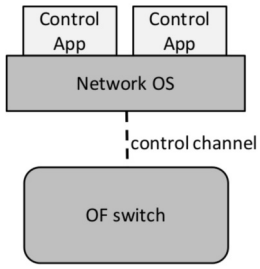


Fig. 1: OpenFlow tool-stack x-ray.

This paper presents the design of OFLOPS-SUME, a NetFPGA-SUME [9] based hardware-software co-design for switch characterization. OFLOPS-SUME builds upon OFLOPS [10], a software SDN testing platform, and OSNT [11], a high-performance traffic generation and measurement platform for the NetFPGA SUME. We introduce the new design of OSNT, previously available only on the NetFPGA-10G board [12], as well as its integration with OFLOPS. We use the proposed architecture to test and characterize both software and hardware OpenFlow switches. To demonstrate the relevance of OFLOPS-SUME, we use it to expose the undesirable impact of the barrier primitive, potential misbehaviors when adding or modifying a batch of rules and the impact of simple packet manipulation operations on the observed switch performance. OFLOPS-SUME can be used with any hardware/software device offering an OpenFlow control plane, including P4-programmable [13] data planes with OpenFlow support. Moreover, by open-sourcing the tool, we invite the community to develop support for additional SDN control planes, such as the P4-runtime [14].

The contributions of the paper can be summarized as follows:

- We describe the architecture of OSNT on NetFPGA-SUME as well as its integration with OFLOPS, to obtain a high performance, 40Gbps and beyond, OpenFlow switch testing and characterization platform.
- Using OFLOPS-SUME, we conduct a performance study of both software and hardware switches and discuss the implications of the different switch profiles on the overall network behavior, policy consistency, scalability and performance.
- We release the code publicly as open source[1]. By providing an open source solution, we promote experiments reproducibility as well as encourage the network community to adapt, evolve, and improve our solution to specific needs.

II. Challenges and Opportunities in Network device testing

Two major drivers for SDN are open APIs and the network management flexibility. These two aspects have incentivised the community to devise protocols and standards for the control of multi-technology and multi-vendor networks [15]. However, implementing them is challenging in practice, due to the wide diversity of the actual support and performance across different vendors.

A good example of how an SDN implementation can impact both network operations and its scalability can be found in the blackholing practice: a defense strategy against Distributed Denial of Service (DDoS) attacks, which blocks malicious traffic by redirecting it into a black hole or null route. Characterizing the network control responsiveness, by quantifying *the latency to update the forwarding state of k flows*, is crucial to understand when exactly a blackhole policy is active. Furthermore, the performance of an SDN implementation can introduce transient policy violations during the deployment of a network policy, due to slow or re-ordered rule updates [16]. To ensure consistent policy enforcement, individual devices must implement a specific rule set at all times, even during policy updates. Rule update ordering in OpenFlow networks can be guaranteed using Barrier messages. However, disambiguation in Barrier message semantics between commercial switches can result in unexpected behaviors [17]. Understanding *how accurately a barrier message represents the data plane configuration status* can thus help in implementing better update strategies that overcome policy inconsistencies. Finally, the deployment of hybrid Xeon and FPGA servers in datacenters [18] enables a wide range of performance improvements, allowing a system to control the trade-off between performance (FPGA) and flexibility/scalability (CPU).

A. Switch Architecture

This section provides a high-level design overview of a generic *switch*. We highlight the physical limits in control and data plane performance and motivate our discussion on performance characterization. Our presentation focuses on top-of-rack (ToR) switch devices and high-performance software switch frameworks, the most common network device types with built-in SDN support.

Switch datapath. The datapath usually consists of four functional blocks. The *input arbiter* module schedules incoming packets to the packet processing threads, for software switches, or to the main processing pipeline, for hardware switches. The main processing pipeline is implemented by the *header parser* and the *lookup* modules. The parser module consists of multiple protocol parsers that extract important header information, used by the lookup module to define the per-packet processing and forwarding policy. The header parsing and lookup modules can recirculate packets through the processing pipeline, to parse nested packet headers or to enable multi-table processing datapaths. Finally, the *output arbiter* module enforces traffic policing, applies outstanding packet modifications and forwards packets to the appropriate output ports. Modern hardware based ToR switches use either an ASIC [19], [20], a FPGA [21] or a Network Processor unit [22] for packet processing acceleration. In all cases, the main bottleneck is the size and access speed of its lookup memory modules. In contrast, software switches rely on the parallelization of modern multi-core CPUs to achieve line-rate performance. In this scenario, the main bottleneck is the number of cores and the clock rate. Furthermore, software switches use PCIe channels to connect the system CPUs with the server NICs. The speed and capacity of the PCIe interconnect and the card driver performance can thus be a bottleneck.

Switch Management. The switch management module is responsible for translating high-level control operations into appropriate data plane configuration. In hardware switches, the switch management module is made of different control agents, each providing support for a specific control function, such as legacy routing protocol or SDN support. Depending on the switch model, to support the computational requirements of the control agents, network vendors typically use a general-purpose low-power management CPU, such as PowerPC SoC and Intel Atom. In the former case, the CPU may become a bottleneck when used by interactive control applications. In contrast, the switch management module in software switch runs a separate thread or process and uses OS inter-process communication mechanisms to configure the state of the packet processing threads. Because the management module runs on the same CPU as the packet processing threads, software switch control channels offer enhanced responsiveness.

B. Datapath memory configuration in modern SDN switches

The packet processing functionality in modern SDN switches relies on a fast-lookup abstraction with support for extensive wildcard matching and reconfigurability. Such an abstraction is based on a match-action architecture [23].

Software switches typically implement the match-action tables using software lookup structures, like tuple space search classifiers [24]. Updating such structures usually has very

[1] https://github.com/oflops-nf

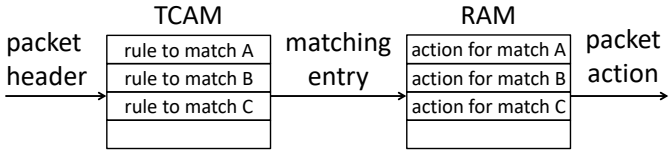


Fig. 2: The two memories used by hardware switch datapaths: (i) TCAMs for flow-match definitions, and (ii) RAMs for forwarding actions.

low latency, but the lookup speed depends on the size and the occupancy of the lookup structure. In contrast, hardware switches implement tables using two different memory subsystems embedded in the hardware. The match definitions are stored in one or more Ternary Content-Addressable Memory (TCAM) modules. TCAMs are optimized for fast lookups, supporting $O(1)$ lookup complexity for any match, irrespective of the lookup key-width or the number of stored match definitions. The output of a TCAM lookup is then used to index a RAM memory module, which stores the actions and statistics associated to the match [25], [26], [23]. Figure 2 shows the interaction between TCAMs and RAM memory in high performance switching engines.

The management module changes the per-packet processing behavior of the switch by manipulating the two memories. Adding a new rule causes an update operation in both TCAM and RAM memories. A modification of the behavior of an existing rule causes an update operation in the RAM only. Unfortunately, the update latency of a single TCAM entry is not constant and depends on the memory design and on the number and structure of the entries already installed [16]. The update latency of a RAM entry is constant and depends solely on the memory type. This update latency mismatch can cause inconsistencies during update operations.

III. OFLOPS-SUME: ARCHITECTURE

A schematic of the OFLOPS-SUME design is depicted in Figure 3. The OFLOPS-SUME architecture consists of a software and a hardware subsystem. The software subsystem runs the core OFLOPS functionality, along with the test of the user. A test contains both the control and data plane logic of the experiment.

While the control logic, i.e., OpenFlow control messages, interacts with the system under test through a commodity network interface card, the data plane logic relies on a NetFPGA-SUME (running the OSNT design) hardware subsystem to fulfill the data plane requirements of the experiment.

We chose to use two separate hardware subsystems for control and data plane channels to balance flexibility and performance. While control plane messages, i.e., OF messages, mainly works at milli-seconds timescales and do not require hardware acceleration, data plane traffic needs high-throughput generation and hardware timestamping to properly characterize switches at 10Gbps and beyond. Control messages are therefore handled by a commodity network interface card, so that future updates of the control message protocol require a software update only. This makes the design flexible and future-proof. On the other hand, data plane traffic is handled using the OSNT hardware acceleration, to guarantee performance.

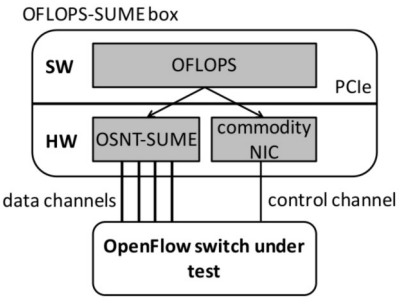


Fig. 3: OFLOPS-SUME design.

The integration of OFLOPS and OSNT on NetFPGA-SUME is achieved through a C library[2], which exposes a traffic generation and capture interface, as well as access to card statistics (*e.g.,* counter for packet captures and drops). The library is designed to exploit as much as possible the OSNT capabilities, i.e., multi-10Gbps packet generation and nanosecond scale hardware timestamping. The user can interconnect the OFLOPS-SUME host with one or more switches in arbitrary topologies, and measure with high precision specific aspects of the network architecture, both of the data and control plane. Next, we briefly describe the design of the OSNT on NetFPGA-SUME hardware (Section III-A) and the OFLOPS software architecture (Section III-B).

A. OSNT: Open Source Network Tester

OSNT is an open–source hardware-based traffic generation and capture system. Originally designed for the research and teaching community, its key design goals were low cost, high-precision time-stamping and packet transmission, as well as scalability. Being open–source, it provides the flexibility to allow new protocol tests to be added. The first prototype implementation was built upon the NetFPGA-10G platform, while the currently supported version relies on the newer NetFPGA-SUME hardware. Figure 4 shows the current OSNT architecture. While the hardware side of OSNT (*i.e.,* NetFPGA-SUME board) provides two independent pipelines for high-precision traffic generation and monitoring, the software provides user APIs to interact with the hardware through the NetFPGA driver.

In the following, we describe the architecture of its main pipelines and the timestamping module which is shared between them.

OSNT Monitor. It provides functions for packet capturing, hardware packet filtering, high precision packet timestamping and high-level traffic statistic gathering. As soon as a packet arrives, a module placed immediately after the physical interfaces, and before the receive queues, timestamps it. In the meantime, internal hardware counters (number of packets/bytes received) exposed to the software are updated accordingly to provide high-level statistics about the traffic.

[2]https://github.com/oflops-nf/nf-pktgencap-lib

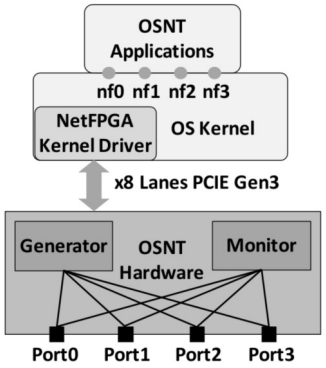


Fig. 4: Architecture of OSNT.

Then, an internal module gathers the relevant information (the 5-tuple[3]) to feed the *TCAM*. The TCAM, implemented using FPGA resources, enables the host to write up to sixteen 5-tuple rules representing the traffic of interest. Only packets that are matched in the TCAM are sent to the software, while all other packets are discarded. Even though our design implicitly copes with a workload of full line-rate per port of minimum sized packets, the input traffic will often exceed the capacity of the host-processing, storage, etc., or may not be of practical interest.

OSNT Generator. It is designed to generate 10Gbps at line-rate, 64B packets, per card interface. Currently, it is able to replay network traffic from PCAP files stored in either the NetFPGA internal or external memories. A set of APIs allow the host to interact with the hardware, to configure the generation parameters. Specifically, upon signaling from the software, the hardware reads its memories and starts the generation process that can follow user-defined timing information, i.e., inter-packet gap, or can strictly depend on the PCAP trace. The generator has an accurate timestamping mechanism, located just before the transmit 10GbE MAC. The mechanism, identical to the one used in the traffic monitoring unit, is used for timing-related measurements of the network, allowing characterization of measurements such as latency and jitter. The timestamp is embedded within the packet alongside a packet counter at a pre-configured location, and can be extracted at the receiver as required.

Timestamping logic. Providing an accurate timestamp to packets is a fundamental objective of OSNT. Packets are timestamped as close to the physical Ethernet device as possible, so as to minimize FIFO-generated jitter and obtain accurate latency measurements. A dedicated timestamping unit, shared between the monitoring and the generation pipelines, stamps packets as they arrive from/to the physical (MAC) interfaces. To minimize overhead while also providing sufficient resolution and long-term stability, OSNT uses a 64-bit timestamp divided into two parts: the upper 32-bits count seconds, while the lower 32-bits provide a fraction of a second with a maximum resolution of about 233ps. Given the 160Mhz board clock, the practical resolution is 6.25ns. Note that accurate timekeeping requires correcting the frequency drift of an oscillator. To this end, OSNT uses Direct Digital Synthesis (DDS), a technique by which arbitrary variable frequencies can be generated using synchronous digital logic [27]. The addition of a stable pulse-per-second (PPS) signal, e.g., derived from a GPS receiver, permits both high long-term accuracy and the synchronization of multiple OSNT elements.

[3]We define 5-tuple as the combination of IP address pair, layer four port pair and protocol.

B. OFLOPS: OpenFlow Operations Per Second

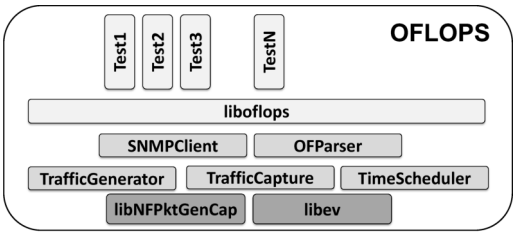


Fig. 5: OFLOPS platform architecture

OFLOPS is a modular software framework that provides a unified API to interact with the data, the control and the management plane of an OpenFlow switch, and log its behavior. By describing flow-level traffic characteristics, users can define test cases which abstracts the complexity to realize the respective functionalities across different platforms, i.e., OSNT, pktgen and libpcap kernel modules. This is done by compiling the experiment into dynamic libraries that are loaded at run-time by the OFLOPS software.

OFLOPS follows a 3-layer architecture, depicted in Figure 5. The bottom layer consists of a set of services enabling interaction with a switch across different control and data channels. Access to the different information and control channels is unified and abstracted by the middle layer of the platform. Finally, the top layer of the OFLOPS architecture consists of measurement modules which implement the interaction scenarios. The system is designed to minimize packet processing overheads and uses a multi-threaded architecture to parallelize event processing, in an effort to minimize packet loss and measurement noise. At boot time, the process initializes five threads, each responsible of managing specific event types, namely (i) traffic generation, (ii) traffic capture, (iii) OF message parsing, (iv) SNMP responses and (v) time events. To avoid synchronization overheads, OFLOPS provides a *non* thread-safe event API. Module developers are responsible of implementing custom synchronization mechanisms within their test logic.

Currently, OFLOPS supports the most widely adopted OF versions: 1.0 and 1.3. The OFLOPS code base contains a set of reference test modules supporting all elementary protocol interactions, like flow table manipulation and traffic monitoring.

C. OFLOPS-SUME in action: using the platform

Listing 1 presents the most important APIs (from now on we call them event handlers) available. In OFLOPS-SUME, each

test needs to comply with the same life cycle: *initialization*, *measurement* and *termination*.

During the initialization stage, OFLOPS-SUME offers the ability to (1) access configuration parameters (`handle_module_init`), (2) bootstrap both traffic generation processes (`handle_traffic_generation`) and capture filters (`handle_pcap_filter`), and (3) finally initialize switch state (`handle_module_start`). In terms of traffic generation, a test must specify ranges of values for each packet field and the inter-packet delay. Similarly, a test can specify wildcard field matching filters, which can be offloaded to the NetFPGA hardware. As an example, the flow insertion module used in § IV, during the initialization stage, configures the traffic generator to emit traffic at a constant rate targeting inserted flows in a round-robin fashion, while the capture filter is configured to accept all packets. Finally, the switch initialization callback inserts the various rules in the switch forwarding table.

During the measurement phase of the experiment, OFLOPS-SUME offers three major monitoring event types: SNMP responses (`handle_snmp_event`), data plane packet interceptions (`handle_pcap_event`) and switch-to-controller OpenFlow message handlers (`handle_of_event_packet_in`). The first one performs asynchronous SNMP polling, the second exploit the OSNT monitoring capabilities to access accurate per packet transmission and receipt timestamps, and the third translates OpenFlow packets to control events.

Finally, the OFLOPS-SUME APIs offer a function to terminate an experiment. Once all monitoring processes are halted, the handler `handle_module_destroy` is invoked to allow the module to gracefully log information and release state. The flow insertion module, for example uses this event to log all packet timestamps in a text file, available for post-processing.

```
// module initialization callback
int handle_module_init(oflops_context *c, char *p)
    ;

//Traffic generation and monitor configuration
int handle_get_pcap_filter(oflops_context *c,
    oflops_channel_name c, cap_filter **f);
int handle_traffic_generation(oflops_context *c);
int handle_cap_filter(oflops_context *c,
    oflops_channel_name c, cap_filter **f);

//Switch initialization
int handle_module_start(oflops_context *c);

//Main thread event handlers
int handle_timer_event(oflops_context *c,
    timer_event *t);
int handle_snmp_event(oflops_context *c,
    snmp_event *s);
int handle_pcap_event(oflops_context *c,
    pcap_event * pe, oflops_channel_name c);
//The OFLOPS API offers a range of OpenFlow
//message handlers similar to the following
int handle_of_event_packet_in(oflops_context *c,
    ofp_packet_in *p);

//Module termination
int handle_module_destroy(oflops_context *c);
```

Listing 1: Sample OFLOPS module API.

Packet Size	OVS-kernel		OVS-DPDK	
	CPU load	Packet Loss	CPU load	Packet Loss
96B	7.5	0.31	1	0.17
128B	6.0	0.24	1	0
256B	2.0	0.22	1	0
512B	0.3	0.01	0.93	0
1024B	0.3	0	0.80	0
1472B	0.3	0	0.56	0

TABLE I: Comparison between OVS-kernel and OVS-DPDK in terms of CPU load and packet loss

IV. Performance characterization

In this section, we present a switch characterization study, using the OFLOPS-SUME platform. In our analysis, we employ three representative OpenFlow switch types based on the OpenVSwitch (OVS) [28] switch platform. Firstly, we employ the software kernel-based OVS (OVS-kernel) switch, highly popular in cloud infrastructures. Secondly, we employ a port of the OVS platform on the DPDK framework (OVS-DPDK)[4], a high-performance zero-copy packet processing framework used extensively to implement network function applications. Finally, we used a white-box hardware switch (EdgeCore AS5812-54X) running an OpenFlow-enabled firmware (PicaOS), a common design choice for ToR switches. The switch offers an Intel Atom quad-core co-processor, which runs a modified OVS version capable to offload data plane processing to a Broadcom ASIC (BCM56854) supporting up to 1.28Tbps throughput and a 32k entry TCAM module.

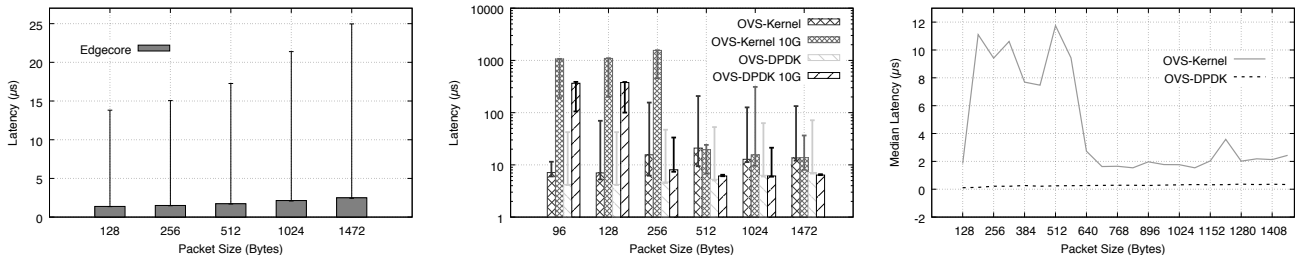
Our measurement study focuses on switch forwarding performance (§ IV-A), the flow table management capabilities, and the consistency characteristics during forwarding policy reconfiguration (§ IV-B). Figure 3 depicts our measurement setup. The OFLOPS-SUME runs on an Ubuntu server (quad-core Intel E5-1603, 16 GB RAM) equipped with a NetFPGA-SUME card and programmed with the OSNT bitfile. The software OVS instances (v2.8.1) run on an Ubuntu server (16-core Intel E5-2630, 32 GB RAM) equipped with a dual-port 10G Intel NIC (82599ES). For all the figures, unless stated otherwise, we present the median value over 20 experiment repetitions alongside the minimum and the 90^{th} percentile.

A. Data plane performance

Figure 6 depicts the forwarding latency of each OpenFlow switch for varying packet sizes at low (100Mbps) and line rate (10Gbps) traffic rates. Figure 6a reports the results for the Edgecore, when no specific packet modification is requested, *i.e.*, only simple forwarding. We observe that packet size variations have a negligible impact on the median latency, which remains stable around $2.5\mu s$. Nonetheless, increasing packet sizes inflate significantly the 90^{th} percentile of the forwarding latency, which is 66% higher for maximum size packets in comparison to 128-byte sized packet.

Software switch forwarding latency, depicted in Figure 6b, is approximately one order of magnitude higher than the hardware switch. In addition, OVS-kernel exhibits higher latency variability than OVS-DPDK. The forwarding latency inflation

[4]http://docs.openvswitch.org/en/latest/intro/install/dpdk/



(a) Packet Forwarding: Edgecore

(b) Packet Forwarding: OVS-kernel and OVS-DPDK

(c) Impact of packet manipulation: OVS-kernel and OVS-DPDK

Fig. 6: Data plane performance analysis for Edgecore, OVS-kernel and OVS-DPDK.

is further exacerbated when switching line rate traffic. Both software solutions inflict two orders of magnitude higher latency for small packets (< 256 bytes), which can be attributed to the extensive packet queuing required to cope with the high packet rate. Nonetheless, OVS-DPDK exhibit better scaling capabilities for large packet sizes, since it avoids unnecessary kernel packet processing. Figure 6c depicts the increase of the median latency caused by a packet manipulation operation[5]. While the EdgeCore (not shown in the figure) and OVS-DPDK remain unaffected, OVS-kernel experiences a median latency increase of $12\mu s$ in the worst case. Such forwarding latencies have been reported to significantly affect the performance of several common datacenter applications [29]. The impact of packet modification is more pronounced for small packets. This can be attributed to the NAPI interrupt mitigation mechanism used by the OVS-kernel switch, which switches between interrupt-driven and polling-based modes depending on the incoming packet rate. In contrast, OVS-DPDK uses exclusively a poll-based packet fetching mechanism, which minimizes the impact of the packet size on forwarding latency, but increases the CPU utilization.

These design differences are observable in Table I, which reports the CPU utilization and packet loss for OVS-kernel and OVS-DPDK[6] when switching line rate traffic. OVS-kernel used up to 8 softirq kernel threads in order to cope with the processing load for small packets, and still experienced significant packet losses. In contrast OVS-DPDK was able to switch line rate traffic using a single core with no packet loss for packet sizes greater than 128 bytes and the system was capable to offer better processing performance that the OVS-kernel configuration. These performance characteristics align with the design philosophy of each platform; OVS-kernel is designed for cloud frameworks and offers a flexible, backwards compatible and low overhead packet processing platform for low and medium traffic rates, but its performance scales poorly at high data rate. OVS-DPDK on the other hand is designed for NFV environments and offer excellent forwarding performance for any traffic rate, but its resource utilization scales poorly.

[5]In the test, we considered a layer 4 port rewriting.

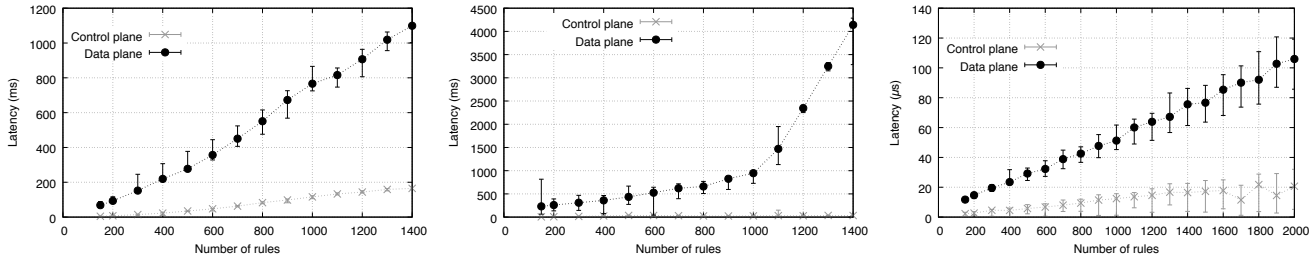
[6]The OVS-DPDK process used 4 CPU cores, which were excluded from the OS scheduler. OVS-DPDK uses the DPDK poll mode driver (PMD) to process packets, which runs an continuous polling loop that saturates the allocated CPU resources. During our experiment OVS-DPDK was running at 100% utilization on the assigned cores and for the CPU utilization measurement we used statistics from the OVS-DPDK process, which report the number of CPU cycles used to process packets from each NIC queue.

B. Control plane performance

Figure 7 shows the results of our control plane performance analysis. We characterize the amount of time needed to insert new flow rules or modify existing ones. In addition, we also focus our attention on assessing the reliability of the barrier primitive, compared to the actual rule installation.

Figure 7a presents the rule update latency of the Edgecore switch for a varying number of rules, using two measurement metrics. The *control plane* latency represents the time between the `FLOW_MOD` message transmission and the reception of the `BARRIER_REPLY` by the switch through the control plane channel (see Figure 3). In contrast, the *data plane* latency measures the time between the `FLOW_MOD` message transmission and the time when at least one packet has been received with the new configuration by OFLOPS-SUME on one of the data channels (see Figure 3). The significant difference between the two measurements (Figure 7a) can be attributed to the behavior of the switch OpenFlow agent. Specifically, we notice that the switch OpenFlow agent transmits the `BARRIER_REPLY` as soon as the flows are received and processed by the co-processor, without checking if the updates have been applied to the ASIC lookup modules. The reported result has a major implication: for a few seconds, control and data plane experience a policy inconsistency which can lead to thousands of mis-routed packets in high performance networks.

Figure 7b shows the performance of the Edgecore switch when we insert an increasing number of rules. Similarly to the flow modifications, the discrepancy between control and data plane is pretty clear. In addition, the increased rule insertion latency, in comparison to flow modifications, can be attributed to the TCAM reordering operation, required to effectively manage TCAM entries. Since our test inserts rule batches with an increasing priority, the switch OpenFlow agent has to re-order the entries every time we add new TCAM rules, to guarantee correct hardware behavior, thus increasing the switch reconfiguration latency. On the contrary, if the new rules were added with a random priority, the agent would have to re-order only some of them, with a consequent decrease of the reconfiguration latency. We do not observe a similar behavior for flow modifications (Figure 7a), since such operations require switch RAM memory writes only and they



(a) Flow modification: Edgecore

(b) Flow addition: Edgecore

(c) Flow addition: OVS-DPDK

Fig. 7: Control plane performance analysis for Edgecore, OVS-kernel and OVS-DPDK.

do not require any TCAM modifications(§II-B). Moreover, it is worth noting the big difference between flow modification and flow addition latencies. The former, for 1400 rules, is approximately 70% faster than the latter. This is again a consequence of the TCAM reordering: a single rule installation can cause many writes in the TCAM and associated RAM.

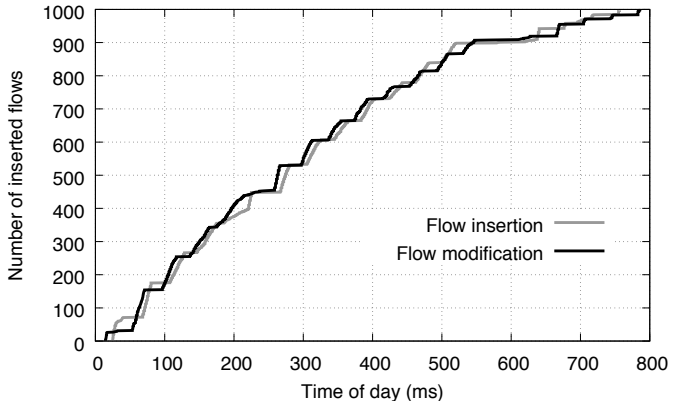
Finally, Figure 7c shows the behavior of rule addition latency for a software OpenFlow switch. Note the almost three orders of magnitude difference with respect to the hardware case. We omit the flow modification results, since they exhibit the same pattern as the flow additions. Effectively, software switches use in-memory data structures to perform the lookup process which have a low update complexity. This makes the rule addition or rule modification processes almost similar: the latency increases linearly with the number of rules changed. In contrast, as discussed in the challenges section (§II), hardware solutions employ TCAMs and RAMs lookup modules which exhibit different update latency characteristics and lead to different behavior during flow additions and flow modifications. In order to get more insights on the behavior of hardware switches during flow modifications, we show in Figure 8a the total number of modified flows over time when running our flow insertion and update measurement modules against the Edgecore switch for 1k flows. Understanding the flow processing behavior of an OpenFlow switch allows SDN developers to improve the control and responsiveness of their applications during large flow table modifications, by controlling the flow transmission rate. From the measurement results, we highlight that flow insertions and flow modifications exhibit a batching behavior, where some flows are inserted during a small time interval followed by a long quiet period (as shown at approximately 530ms). This behavior can be attributed to the design of the underlying ASIC driver, which batches flow updates in order to improve the overall completion time. Similar performance characteristics have been observed with similar hardware switches for `PACKET_IN` messages [30], [31].

Update policy consistency has been an important challenge for OpenFlow [32] switches. Unlike traditional control protocols, OpenFlow offers direct control of the switch forwarding table and thus is prone to access-control policies violations and routing anomalies (e.g. blackholes). Figure 8b explores policy consistency during large flow table updates (1k flows), by comparing the desired flow ordering and the real one obtained through data-plane measurements. From the figure we note that they are not consistent. Indeed, in an ideal scenario where rules are inserted following the original order, a straight line would have been expected. On the contrary, our measurements show that rules are not only inserted in batches but also follow a random ordering. This can be explained by the architecture of the OVS vswitchd software, used by the PicOS firmware on the Edgecore switch. All flow table entries are stored by default in the vswitchd daemon running in the switch co-processor and flows are installed in the control plane only if the ASIC propagates to the daemon a matching packet. After that, the vswitchd daemon will install the flow in the ASIC TCAM. Nonetheless, during large policy updates the interconnect between the ASIC and the co-processor is saturated and packets are pushed to the co-processor in a best effort fashion, thus randomizing the flow insertion order.

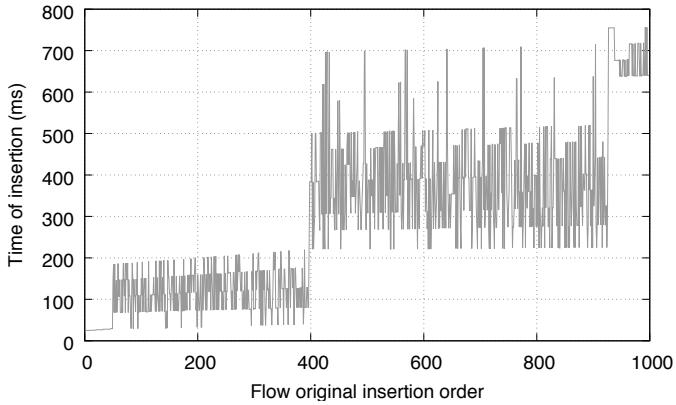
V. Related Work

The SDN community has developed a wide range of validation and testing tools. OFtest [33] is a standards conformity evaluation framework for OpenFlow switches. The suite offers tests to verify the conformance of a switch implementation against a specific OpenFlow standard. In parallel, Cbench [6], OFLOPS [10] and OFLOPS-TURBO [34] are focused primarily on the performance side. Cbench evaluates the performance of a controller by emulating a number of switches generating `PKT_IN` messages and measuring its responsiveness. The OFLOPS and OFLOPS-Turbo platforms concentrate on switch performance evaluation. While OFLOPS is a pure software solution limited to 1 GbE devices and supporting the 1.0 OpenFlow protocol version, OFLOPS-Turbo integrates an old version of NetFPGA to support higher throughput. However, its limited hardware resources make the solution unattractive to enable the support of new OpenFlow versions as well as more refined data plane tests.

Switch characterization studies have also aimed to identify the impact of SDN technologies on network performance. Bianco *el al.* [35] conducted the first data plane performance study of software OpenFlow switches. In their study, they highlighted the performance improvements of Open vSwitch over traditional Linux bridging, and showed the performance improvement of NAPI interrupt mitigation on network performance. Starting from their lessons learned, in this paper, we also show the impact of emerging fast I/O frameworks,



(a) Flow insertion number timeseries.



(b) Comparison on desired and real flow insertion ordering.

Fig. 8: Flow insertion and modification analysis .

such as DPDK, on software switching performance. Huang *el al.* [30] benchmarked OpenFlow-enabled switches and illustrated how their implementation can significantly impact data plane latency and throughput. They also presented a measurement methodology and emulator extension, to reproduce these control path performance characteristics, restoring the fidelity of emulation. While the authors focused primarily on vendor-specific variations in the control plane, in this paper, we propose a benchmark for both control and data plane, taking advantage of the OSNT system. Tango [36] is a controller design capable of evaluating the performance profile of a network switch. The controller uses control plane traffic injection and capture during a training phase, to evaluate the performance characteristics of a network switch. The controller uses these results at run-time to adapt the policy deployment strategy. Unfortunately, the precision of the resulting system remains poor, as the control channel offers pretty limited performance and precision guarantees. Furthermore, Curtis *el al.* [37] discussed the switch specific limitations to support OpenFlow functionality in large network deployments. Their measurement study highlighted the impact of the channel between the co-processor and the ASIC in policy reconfiguration. Finally, Jarschel *el al.* [38] employed a DAG-based network environment to measure the flow manipulation performance of OpenFlow switches, and presented a markov model for the forwarding speed and blocking probability of OpenFlow devices.

VI. Conclusions

We presented OFLOPS-SUME, a performance characterization platform for OpenFlow switches. OFLOPS-SUME combines the advanced hardware capabilities of OSNT, implemented on the NetFPGA-SUME, with the extensibility of the OFLOPS software framework to enable switch characterization at 40Gbps and beyond.

Using OFLOPS-SUME, we evaluated both software and hardware switches. The performed measurement campaign exposes the sometimes undesirable impact of barrier primitives, potential misbehaviors when adding or modifying a batch of rules and the cost in terms of added latency of simple operations, such as packet modification. OFLOPS-SUME can be used with any hardware/software device that uses OpenFlow as a control plane. Thus, it can be also be exploited by highly programmable devices, such as P4-enabled ones, as long as they are managed with an OpenFlow control plane. We release the code publicly as open source to promote experiments reproducibility, as well as to encourage the network community to evolve our solution.

Acknowledgment

This research is (in part) supported by the UKs Engineering and Physical Sciences Research Council (EPSRC) under the EARL (EP/P025374/1) and the TOUCAN (EP/L02009/1) projects.

References

[1] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," in *Proceedings of IEEE, Volume: 103, Issue: 1*. IEEE, 2015.

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-deployed Software Defined Wan," in *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2013.

[3] M. Kobayashi, S. Seetharaman, G. Parulkar, G. Appenzeller, J. Little, J. Van Reijendam, P. Weissmann, and N. Mckeown, "Maturing of OpenFlow and Software-defined Networking Through Deployments," in *Computer Network, Volume: 61*. Elsevier, 2014.

[4] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A NICE Way to Test Openflow Applications," in *Networked Systems Design and Implementation (NSDI)*. USENIX, 2012.

[5] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A Flexible OpenFlow-Controller Benchmark," in *European Workshop on Software Defined Networking (EWSDN)*. IEEE, 2012.

[6] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," in *Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*. USENIX, 2012.

[7] D. Turull, M. Hidell, and P. Sjodin, "Performance evaluation of OpenFlow controllers for network virtualization," in *High Performance Switching and Routing (HPSR)*. IEEE, 2014.

[8] Y. Zhao, L. Iannone, and M. Riguidel, "On the Performance of SDN Controllers: A Reality Check," in *Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, 2015.

[9] N. Zilberman, Y. Audzevich, A. G. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as Research Commodity," in *Micro, Volume: 34, Issue: 5*. IEEE, 2014.

[10] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An Open Framework for Openflow Switch Evaluation," in *Passive and Active Measurement (PAM)*. Springer-Verlag, 2012.

[11] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. G. Covington, M. Bruyere, N. McKeown, N. Feamster, B. Felderman, M. Blott, A. W. Moore, and P. Owezarski, "OSNT: Open Source Network Tester," in *Network, Volume: 28, Issue: 5*. IEEE, 2014.

[12] M. Blott, "FPGA Research Design Platform Fuels Network Advances," in *Xcell, Issue: 73*. Xilinx, 2010.

[13] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," in *SIGCOMM Computer Communication Review, Volume: 44, Issue: 3*. ACM, 2014.

[14] P4 Language Consortium, "P4runtime," https://p4.org/p4-runtime.

[15] M. Casado, N. Foster, and A. Guha, "Abstractions for Software-defined Networks," in *Communications of the ACM, Volume: 57, Issue: 10*. ACM, 2014.

[16] H. Chen and T. Benson, "The Case for Making Tight Control Plane Latency Guarantees in SDN Switches," in *Symposium on SDN Research (SOSR)*. ACM, 2017.

[17] J. H. Han, P. Mundkur, C. Rotsos, G. Antichi, N. Dave, A. W. Moore, and P. G. Neumann, "Blueswitch: enabling provably consistent configuration of network switches," in *Architectures for Networking and Communications Systems (ANCS)*. ACM/IEEE, 2013.

[18] P. K. Gupta, "Xeon+FPGA Platform for the Data Center," https://www.ece.cmu.edu/~calcm/carl/lib/exe/fetch.php?media=carl15-gupta.pdf, online; accessed August 2017.

[19] "Juniper high-performance data center switches," https://www.juniper.net/uk/en/products-services/switching/qfx-series/, online; accessed February 2018.

[20] "Cisco data center switches," https://www.cisco.com/c/en/us/products/switches/data-center-switches/index.html, online; accessed February 2018.

[21] "Corsa Technology," www.corsa.com/solutions/, online; accessed February 2018.

[22] "NoviFlow Top-Of-Rack switching," https://noviflow.com/tor/, online; accessed February 2018.

[23] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN," in *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2013.

[24] V. Srinivasan and G. Varghese, "Fast Address Lookups Using Controlled Prefix Expansion," in *Transactions on Computer Systems, Volume: 17, Issue: 1*. ACM, 1999.

[25] R. Giladi, "Network Processors: Architecture, Programming and Implmentation." Elsevier, 2008.

[26] N. Stringfield, R. White, and S. McKee, "Cisco Express Forwarding." Cisco Press, 2013.

[27] P. Saul, "Direct digital synthesis," in *Circuits and Systems Tutorials*. IEEE Press, 1996.

[28] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *Networked Systems Design and Implementation (NSDI)*. USENIX, 2015.

[29] N. Zilberman, M. Grosvenor, D.-A. Popescu, N. Manihatty-Bojan, G. Antichi, M. Wojcik, and A. W. Moore, "Where Has My Time Gone?" in *Passive and Active Measurement (PAM)*. Springer, 2017.

[30] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity Switch Models for Software-defined Network Emulation," in *Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2013.

[31] D. Pediaditakis, C. Rotsos, and A. W. Moore, "Faithful reproduction of network experiments," in *Architectures for Networking and Communications Systems (ANCS)*. ACM/IEEE, 2014.

[32] N. P. Katta, J. Rexford, and D. Walker, "Incremental Consistent Updates," in *Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2013.

[33] P. Floodlight, "oftest," http://www.projectfloodlight.org/oftest/.

[34] C. Rotsos, G. Antichi, M. Bruyere, P. Owezarski, and A. W. Moore, "OFLOPS-Turbo: Testing the next generation OpenFlow switches," in *International Conference on Communications (ICC)*. IEEE, 2015.

[35] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: Data plane perfommance," in *Internation Conference on Communications (ICC)*. IEEE, 2010.

[36] 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 *Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. ACM, 2014.

[37] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2011.

[38] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *International Teletraffic Congress (ITC)*. IEEE, 2011.

Rémi Oudin is a master student at École Normale Supérieure Paris-Saclay, France. His interests focus on network performance and measurements, Software Defined Networking and reliability.

Gianni Antichi received a Ph.D. degree in information engineering from the University of Pisa (2011). He is currently a Lecturer at Queen Mary University of London, UK. His research spans the area of reconfigurable hardware, high speed network measurements and Software Defined Networking.

Charalampos Rotsos received his Ph.D. degree in computer science from the University of Cambridge (2014). He is currently a Lecturer with the Lancaster University. His research explores areas of network experimentation, network programmability and orchestration, and operating systems.

Andrew W. Moore the reader in systems at the University of Cambridge, U.K., where he jointly leads the Systems Research Group working on issues of network and computer architecture with a particular interest in latency-reduction. His research contributions include enabling open network research and education using the NetFPGA platform; other research pursuits include low-power energy-aware networking, and novel network and systems data center architectures.

Steve Uhlig received a Ph.D. degree in applied sciences from the University of Louvain (2004). He is currently a Professor of Networks at Queen Mary University of London. His research interests are focused on the large-scale behaviour of the Internet, Internet measurements, software-defined networking, and content delivery.