Transition to SDN is HARMLESS: Hybrid Architecture for Migrating Legacy Ethernet Switches to SDN

Levente Csikor, Márk Szalay, Gábor Rétvári, Gergely Pongrácz, Dimitrios P. Pezaros, *Senior Member, IEEE, ACM*, and László Toka

***Abstract*— Software-Defined Networking (SDN) offers a new way to operate, manage, and deploy communication networks and to overcome many long-standing problems of legacy networking. However, widespread SDN adoption has not occurred yet due to the lack of a viable incremental deployment path and the relatively immature present state of SDN-capable devices on the market. While continuously evolving software switches may alleviate the operational issues of commercial hardware-based SDN offerings, namely lagging standards-compliance, performance regressions, and poor scaling, they fail to match the cost-efficiency and port density. In this paper, we propose HARMLESS, a new SDN switch design that seamlessly adds SDN capability to legacy network gear, by emulating the OpenFlow switch OS in a separate software switch component. This way, HARMLESS enables a quick and easy leap into SDN, combining the rapid innovation and upgrade cycles of software switches with the port density and cost-efficiency of hardware-based appliances into a fully dataplane-transparent and vendor-neutral solution. HARMLESS incurs an order of magnitude smaller initial expenditure for an SDN deployment than existing turnkey vendor SDN solutions while, at the same time, yields matching, or even better, data plane performance for smaller enterprises.**

***Index Terms*— SDN, migration, OpenFlow, switch design.**

I. Introduction

SDN offers a radical break with traditional ways of building, operating and managing networks. By the physical and logical separation of the network control plane from the packet processing functionality, SDN exposes new levels of *abstraction* to the operator, hiding the specifics of the underlying data plane technologies from the network control behind a standardized southbound interface, and unprecedented network and service *programmability*, since the network is now controlled by an adaptable software functionality via an open API. Migration to SDN architecture improves network operations by eliminating the need for box-by-box management and troubleshooting, eases to create network functions and services due to the flexibility of global network view in the centralized control plane, and allows operators to easily buy into new models for network management and operations, like automated orchestration, on-the-fly service chaining, and multi-tenancy support in "as-a-service" schemes [1]–[6].

Despite the fact that many large corporations (Google [7], Microsoft [8], Amazon [9]) and telcos (Orange, Verizon, Deutsche Telekom [10]) have already gained significant foothold in SDN, *smaller businesses, campus network operators, and service providers* seem reluctant to adopt it *en masse* [11]–[13]. In fact, enterprises without substantial in-house expertise and select IT staff ("organizations that aren't called Google" [2]) face significant business, economic, and technical deployment barriers, since migration to SDN requires a nontrivial amount of forward planning, an extensive investigation of vendor offerings and device options, and a fairly radical change in the mental model, producing a typical chicken and egg problem [1]–[6], [14].

First, there is a broad selection of *SDN migration strategies* to choose from, each involving different cost, performance, service availability, management, and security trade-offs that need to be carefully assessed in advance [1]. Incremental deployment strategies may offer the smoothest upgrade path and the least interference with daily network operations [15], yet managing heterogeneous network architectures may prove challenging due to the nontrivial ways the legacy and SDN control planes can interact [3], [4], [6]. Jumping outright to

Manuscript received October 6, 2018; revised March 22, 2019, June 20, 2019, and September 9, 2019; accepted December 2, 2019; approved by IEEE/ACM Transactions on Networking Editor S. Mascolo. The work was supported in part by the UK Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/N033957/1 and Grant EP/P004024/1, in part by the European Cooperation in Science and Technology (COST) Action CA 15127: RECODIS – Resilient communication and services, and in part by the National Research, Development and Innovation Office (NKFIH) under the research and development project in Hungarian-Korean cooperation (project identifier: 2018-2.1.17-TÉT-KR-2018-00012), and under research projects no. FK 128233 and PD 121201, financed under the FK_18 and PD_16 funding schemes, respectively. *(Corresponding author: László Toka.)*

L. Csikor was with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, 1117 Budapest, Hungary. He is now with the Department of Computer Science, National University of Singapore, Singapore 119077 (e-mail: levente.csikor@gmail.com).

M. Szalay is with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, 1117 Budapest, Hungary (e-mail: szalay@tmit.bme.hu).

G. Rétvári is with the MTA-BME Information Systems Research Group, 1117 Budapest, Hungary, with the TrafficLab, Ericsson Research, 1117 Budapest, Hungary, and also with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, 1117 Budapest, Hungary (e-mail: retvari@tmit.bme.hu).

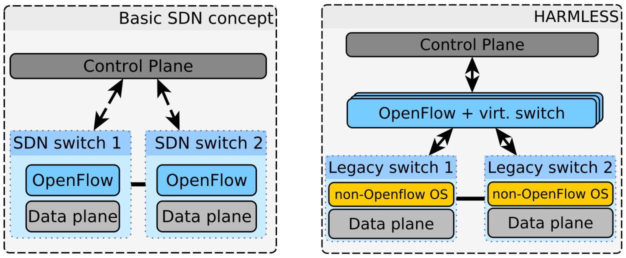
G. Pongrácz is with the TrafficLab, Ericsson Research, Ericsson AB, 1117 Budapest, Hungary (e-mail: gergely.pongracz@ericsson.com).

D. P. Pezaros is with the School of Computing Science, University of Glasgow, Glasgow G12 8QQ, U.K. (e-mail: dimitrios.pezaros@glasgow.ac.uk).

L. Toka is with the MTA-BME Network Softwarization Research Group, 1117 Budapest, Hungary, with the MTA-BME Information Systems Research Group, 1117 Budapest, Hungary, and also with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, 1117 Budapest, Hungary (e-mail: toka@tmit.bme.hu).

Digital Object Identifier 10.1109/TNET.2019.2958762

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) SDN with OpenFlow (b) HARMLESS

Fig. 1. HARMLESS: SDN with an additional level of separation inside forwarding elements.

full-blown SDN by swapping all legacy network gear to SDN-capable devices overnight may mitigate this pain factor, but greenfield migration is hardly an option for small businesses due to the huge capital expenditure, the flag-day deployment, the induced service downtime, and the amount of necessary planning and testing. *A lightweight SDN migration story that would combine the smoothness and reversibility of incremental upgrades with the swiftness and transparency typical to greenfield deployments is still largely missing* [16], [17].

Second, there is the *paradox of choice* inherent to the booming and immature state of the SDN market today, with a breadth of vendor SDN offerings, turnkey solutions, technology trends, and marketing hype, that a less informed operator may find very challenging to explore and evaluate [1], [10]. For a starter, to obtain an SDN-enabled appliance a network operator has essentially two nontrivial choices: buying *commercial off-the-shelf (COTS) and white-box*[1] hardware switches or relying on, possibly already purchased and deployed, *general-purpose servers and running a virtual software switch on top* (e.g., Open vSwitch, OVS [18]). Thanks to the use of special-purpose ASICs and network processors, hardware SDN network gear has traditionally been praised for providing high port density at a reasonable price, albeit notorious for lacking standards-compliance, performance regressions, and scalability [16], [17], [19]. Software SDN switches, on the other hand, struggle to match the port density of hardware switches due to the physical space limits of blades and the steep price of multi-port NICs, but at the same time excel at programmability, extensibility, and standards-compliance [18]. While, thanks to recent advances in softswitch design and implementation [18], [20]–[24] the performance tax of software switching has greatly decreased, *programmability and port density are still competing, if not mutually exclusive, goals in current SDN networking equipment*.

We propose HARMLESS, a *Hybrid ARchitecture for Migrating Legacy Ethernet Switches to SDN*, to foster SDN migration for smaller enterprises. HARMLESS leverages the current trend for "software-defined-everything", but takes this idea to the extreme: it applies an *additional* level of abstraction on top of the conventional control plane–data plane separation by *further decoupling the packet processing hardware from the switch's operating system*, which are today integrated in COTS devices in a single box, *and implementing the OpenFlow (OF) OS in a dedicated software switch* (see Fig. 1). This makes it possible to add SDN capability to plain Ethernet switches, or to any legacy network device, through bypassing the legacy switch OS. Thanks to the additional level of virtualization, *HARMLESS realizes a delicate sweet spot between hardware and software SDN switching*. In particular, *it combines the advantages of software and hardware switching, whereas the hardware component delivers the high port density and raw packet processing functionality, and the softswitch adds programmability, adaptability, and standards-compliance*. Using extensive measurements with a HARMLESS prototype, we show that the benefits of HARMLESS are realized with no significant impact on raw packet processing performance, latency, and dataplane transparency.[2]

From an economical point of view, *HARMLESS offers a viable migration strategy to smaller enterprises.* Since HARMLESS leverages the *existing* network infrastructure it offers distinct price advantages over SDN alternatives available on the market (see details in Sec. II). Crucially, in cases where commodity switches and bare-metal servers for running the OpenFlow (OF) component are readily available, like in smaller private clouds, enterprise networks, research environments, *HARMLESS makes it possible to get into SDN instantly, incurring zero expenditure for a partial or even a complete deployment.* And even if equipment must be purchased anew, HARMLESS can save up to an order of magnitude investment. In a broader perspective, HARMLESS sheds a fresh new light on the ages-old, and often highly contentious, "hard switch vs softswitch" debate and presents an interesting new dimension in switch architectures [25]–[30].

The paper is organized as follows. In Sec. II, we discuss recent market trends in SDN switching, in Sec. III we present the HARMLESS architecture, in Sec. IV we provide a technical and economic evaluation, in Sec. V we summarize related work, and finally in Sec. VI we conclude our work.

II. SDN Switches: Hardware or Software?

The "hard vs softswitch" debate is one of the most disputed one in networking realms and still seems far from concluded [23], [25]–[29], [31]. Therefore, before digging into the architectural details, we give an overview on recent market trends in SDN hardware and software switching appliances, and we show a detailed price analysis to motivate HARMLESS.

On the surface, hardware switches seem unbeatable for packet mangling; packet switching, being a massively parallel and repetitive process, can be accelerated to a large extent using purpose-built ASICs and network processors, while resorting to the CPU has always been considered the "slow path" due to limited memory and I/O bandwidth. Furthermore, a hardware switch can host 48 or even 96 high-speed ports in a single housing, while software switches are limited to some NICs due to the specifics of the form factor. Finally, a TCAM can do packet classification much faster than a general purpose CPU, especially in classification intensive workloads [28]–[30], while software-based packet classification is still an active research field [18], [24], [29], [32]–[34]. Software switches, on the other hand, are easier to upgrade, program, and extend, at shorter time scales than hardware switches, where the developing and bringing a new ASIC to market is resource-intensive and time consuming: requiring an average of four

[1] White-box switches are generically branded switches with no default network operating system.

[2] The performance is upper bounded by the used software switch.

years [35]. Moreover, the performance of general purpose CPUs increases each year, scaling with Moore's law.

The state-of-the-art in SDN switching is more complex.

Standards-Compliance

Being a relatively new technology, the quality of SDN support in COTS devices is variable at best. With many vendors piggybacking OF capabilities on top of an already established switch product line, users are widely complaining about missing OF features (even as elementary as IP address rewrite [16]), switch control plane performance and delays in updating the data plane, atomic flow modification commands not being applied atomically or at all [19], [36], etc. In fact, the need to work around bugs, missing features, and proprietary extensions in COTS SDN switches can lead to operators being locked to a specific vendor. Softswitches, on the other hand, are standards-compliant and rapidly evolve with new standards, and receive bug fixes fast [18], [24].

Scalability

Deploying a new OF agent on a switch does not change the forwarding hardware overnight. Since the TCAM space available for packet classification is limited and since OF rules consume more TCAM space than, say, IPv4 routing table entries, COTS SDN switches rapidly run out of flow table space as the workload increases [17]. Consequently, most COTS switches that are affordable for smaller enterprises support only a couple of hundred (1^{st} gen. "rebranded" devices) or thousand (2^{nd} gen. devices with reasonable price) flow entries in TCAM and, optionally, a handful of additional flows in software (to supplement this imperfection to a certain extent), subject to various intricate limitations on the number of fields that can appear in a rule, rule length, etc. (See Section IV for concrete examples.). Note that some newer generation high performance OF switches [37]–[39] offering high performance with up to 1 million flow rules are available in the market, however not just their costs hinders smaller enterprises to obtain one, but due to the way ASICs are designed an arbitrary forwarding pipeline cannot be applied without fulfilling certain requirements [40] (e.g., wire-speed VLAN handling can only be done in table 0). Therefore, in this work we focus on the great majority of 1^{st} and 2^{nd} generation SDN switches.

Contrarily, softswitches have unlimited flow space in memory but only a subset, CPU caches, can be used efficiently [24].

Performance

The performance penalty of soft SDN switching has effectively diminished recently, thanks to advances in user-space network stacks [20], multi-threaded switch designs [21], [22], hierarchical flow caching [18], and custom-compiled OF datapaths [24]; a modern SDN softswitch easily handles 10G line rate in a single core [29], even with minimum-sized packets [24] profusely meeting the demands of smaller networks. Moreover, as a hardware SDN switch runs out of limited TCAM space it falls back to pure software-based packet forwarding on the slow path, after which point softswitches, designed for CPU-based forwarding from the outset, have clear performance edge [16], [17], [19].

TABLE I

MINIMUM PRICES OF COTS/WHITE-BOX SWITCHES SUPPORTING OPENFLOW VERSION 1.0 OR 1.3 AS OF 2019

Vendor	1G ports		10G ports		40G ports
	24	48	24	48	24-32
Accton EdgeCore		AS4610 $2,200		AS5610 $4,800	AS6812 $7,300
Dell		S3048 $2,500		S4048 $5,600	
HPE / Aruba	5130 $2,400	3800 $7,700	5920 $10,000	5940 $16,800	5930 $16,200
Huawei	S5720 $1,000		S6720 $2,900	S6720 $3,900	
Netberg				AURORA 420 $4,000	
Quanta		T1048 $2,200		T3048 $4,100	T5032 $7,200
Minimum	$1,000	$2,200	$2,900	$3,900	$7,200

Cost-Efficiency

Current network gear costs vary in a wide range, making it challenging not only to compare actual prices but also to predict future price tags. Worse yet, many vendors offer significant discounts for even medium size bulk equipment orders, which is difficult to take into consideration in a cost model. This is the reason why many traditional vendors, e.g., Broadcom, Cisco, Extreme Networks, or NEC, either do not publicly disclose list prices or offer only relatively high list prices, which are then subject to discounts for bulk purchases.

In order to account for these unknowns, our cost model is intentionally simplistic and conservative. In particular, we take the position of a small enterprise that initiates a moderate sized green-field network deployment. Correspondingly, our cost model considers the lowest priced switches available at the time of writing and we did not consider any vendor discounts.

The great majority of COTS and white-box SDN switch market options come with 24 or 48 ports at 1G (or 10G), supplemented with 2–4 uplink ports operating at 10G (or 40G, respectively) in 1U or 2U form-factor. The least expensive offers come at $2,200$ and $3,900$ at the moment; see a non-exhaustive market survey in Table I.[3] Based on these considerations, the following formulas give a conservative estimate for the CAPEX of a hardware switch deployment with a total of x ports:

$$C_{HW}^{1G} = \$2200 \left\lceil \frac{x}{48} \right\rceil, \quad C_{HW}^{10G} = \$3900 \left\lceil \frac{x}{48} \right\rceil \; .$$

In case of softswitches, the main CAPEX factor is purchasing servers with a sufficient number of NICs and CPUs. We consider x86-based 1U servers at a bulk price of $\$1,400$ on average, including the motherboard with 1 CPU, $4\times$1G built-in ports, 3 PCIe (3.0 $\times 8$ or $\times 16$) slots, memory, disk, power, etc. A server can host up to 3 additional NICs, costing $\$100$ for $4\times$1G (Intel i350), $\$400$ for $4\times$10G (Intel X710), and $\$500$ for $2\times$40G (Intel XL710), which total up to 16 ports per server at $1G$, 12 ports at 10G, and 6 ports at 40G. Note that when a single server cannot provide the required port density another server must be purchased. Furthermore, in most cases the third PCIe slot is hardwired to the second CPU socket, therefore a single CPU server can host up to 12 pieces of 1G (8 at 10G, 4 at 40G) ports; for more ports per server a second CPU must be installed (hence the last negative term in the below formulas, where the variable $\#extraCPU$ indicates in both the 1G and

[3] Sources: bm-switch.com, router-switch.com and whiteboxswitch.com.

10G cases whether the last server is sufficient to serve the rest of the ports w/o an additional CPU). The price of a server CPU ranges between $\$200$ and $\$7,000$ depending on the CPU class, cache size, clock rate, and power consumption; we calculate with the price of a 6-core Intel E5-2620v3 CPU at $\$400$.

With this in mind, the following formulas estimate the CAPEX of a software switch deployment with x ports:

$$\begin{aligned} C_{SW}^{1G} &= \$1400 \left\lceil \frac{x}{16} \right\rceil + \$100 \left(\left\lceil \frac{x}{4} \right\rceil - \left\lceil \frac{x}{16} \right\rceil \right) \\ &\quad + \$400 \left(\left\lceil \frac{x}{16} \right\rceil - \#extraCPU_{SW}^{1G} \right) \\ C_{SW}^{10G} &= \$1400 \left\lceil \frac{x}{12} \right\rceil + \$400 \left\lceil \frac{x}{4} \right\rceil \\ &\quad + \$400 \left(\left\lceil \frac{x}{12} \right\rceil - \#extraCPU_{SW}^{10G} \right) \end{aligned}$$

$$\#extraCPU_{SW}^{1G} = \begin{cases} 1, & \text{if } 0 < (x \bmod 16) < 13 \\ 0, & \text{otherwise} \end{cases}$$

$$\#extraCPU_{SW}^{10G} = \begin{cases} 1, & \text{if } 0 < (x \bmod 12) < 9 \\ 0, & \text{otherwise} \end{cases}$$

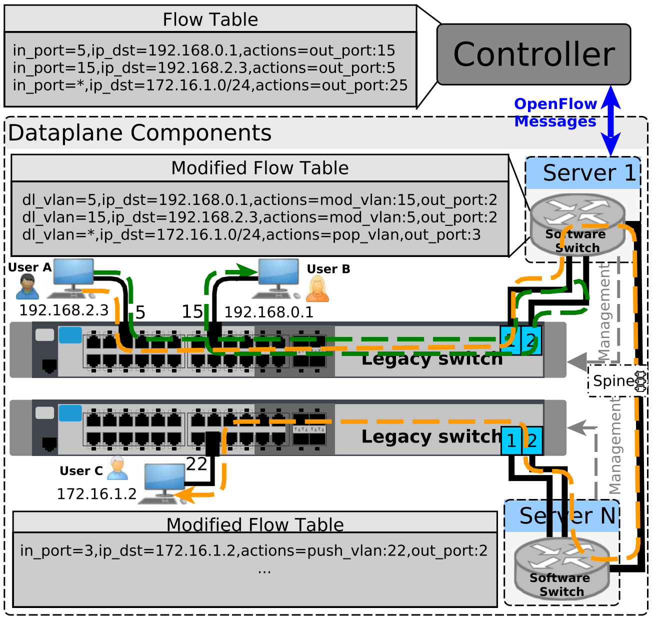
For the sake of simplicity, in the price analysis we do not account for energy consumption, cooling, cabling, rack space occupancy, etc., and we did not take into account the forwarding and trunk port availability a typical COTS device offers; we return to further CAPEX/OPEX issues later.

In summary, current market trends suggest that *hardware SDN switches provide high port density at moderate prices but lag behind software switches in scalability, standards-compliance and programmability*. Next, we present a new SDN switch design called HARMLESS a cost-efficient middle-ground between the two extremes.

III. The HARMLESS Architecture

How to magically turn a legacy Ethernet switch into an OpenFlow-speaking one? This would require to open up what is traditionally a closed black box and substitute the legacy switch OS with an SDN-capable one, something that has proved notoriously difficult to do so far. Instead, in order to preserve backward compatibility, we adopt a more viable approach in HARMLESS by *extending* the "Tagging and Hairpinning" technique (also called *anything-on-a-stick* [41], [42], *distributed switch* design [43], or VEPA [44]/VNTag [45]), originally advocated for hypervisor switches by Cisco and HP, to the general context of SDN [29].

The *essence of "Tagging and Hairpinning" is* to offload VM-to-VM communication from the hypervisor to the first hop switch, i.e., to "outsource" (some parts of) the switching mechanism *to achieve highly improved performance*. When a VM sends a packet it is marked by a unique VLAN id ("tagging") and forwarded to the access switch, which will then do a forwarding/policy lookup to decide whether to loop the packet back to another VM, resident on the hypervisor, in which case it is marked with the unique VLAN id of the target VM ("hairpinning"), or send it further along the data center fabric, or drop it right away. The rationale for this technique is that packet processing is done on efficient special purpose hardware at the first hop switch instead of a potentially less powerful hypervisor switch, while the downsides are doubling bandwidth utilization and increased latency. One of the main contributions of this paper is the observation that, when cast in the general context of SDN switching, *the "Tagging and Hairpinning" technique can be used to provide SDN capability to legacy switches*. Moreover, the way HARMLESS implements this technique yields a uniquely cost-efficient organization of packet processing functionality and forwarding intelligence, and presents an appealing incremental SDN deployment path. Last but not least, HARMLESS *preserves backward compatibility*, which is an all-important factor (not just for network operators) when evaluating viable migration strategies.



```
in_port=5,ip_dst=192.168.0.1,actions=out_port:15
in_port=15,ip_dst=192.168.2.3,actions=out_port:5
in_port=*,ip_dst=172.16.1.0/24,actions=out_port:25
```

```
dl_vlan=5,ip_dst=192.168.0.1,actions=mod_vlan:15,out_port:2
dl_vlan=15,ip_dst=192.168.2.3,actions=mod_vlan:5,out_port:2
dl_vlan=*,ip_dst=172.16.1.0/24,actions=pop_vlan,out_port:3
```

```
in_port=3,ip_dst=172.16.1.2,actions=push_vlan:22,out_port:2
...
```

Fig. 2. HARMLESS: the strawman's approach.

A. High Level Overview

So how to achieve that the commodity switch still realize the required forwarding program? The idea of HARMLESS is to extend the "Tagging and Hairpinning" technique: let the commodity switch tag each packet with a unique VLAN id that identifies the access port it was received from, forward the tagged packet to the software switch along the trunk-port–softswitch interconnect (the *uplink*) to enforce the OpenFlow forwarding and security policies, and send the packet back to the commodity switch via another uplink "hairpinned", i.e., tagged with the unique VLAN id of the proper outgoing port as per the specified flow table. (If the packet was already VLAN-tagged, the commodity switch can use VLAN Q-in-Q tunneling [46] or any other tagging scheme.) In order to reach this, a strawman's approach would suggest that the controller needs to program a slightly modified flow table into the software switch, whereas the "match on ingress port X" rules are converted to "match on VLAN id X" rules and the "output to port Y" actions are rewritten to "modify VLAN id to Y and output to default port" actions; later, we show transparent HARMLESS setups that make such modification of the forwarding program unnecessary.

In our particular example (see Fig. 2), suppose that an operator is given a manageable Layer-2 (L2) 802.1Q Ethernet switch with free high-speed trunk ports, a general purpose server that has spare NICs and available capacity to run an

OpenFlow vswitch, and adequate cabling, backplane capacity, or other means for interconnecting the switches' trunk ports with the softswitch NICs. Suppose further that a host with IPv4 address `192.168.2.3` connected to port 5 on the commodity switch wishes to send packets to another host with address `192.168.0.1` connected to port 15 on the same switch, and suppose that a security policy is in place according to which these two hosts are permitted to exchange traffic only between each other. Finally, suppose that the operator wants to control the switch through OpenFlow and so wishes to program the forwarding behavior as given by the *Flow table* in Fig. 2 (disregarding the standard complexities of IP forwarding, like TTL handling and MAC resolution, for now). This would handle communication between the two hosts adequately, except that it is impossible to control the commodity switch through OpenFlow due to the black box nature of legacy COTS appliances. However, upon receiving a packet on port 5 from the first host, the commodity switch would tag it with a unique VLAN id 5, and send it along the uplink. The softswitch at *Server 1* in turn identifies the original input port based on the VLAN id and makes sure that the originating host is allowed to communicate with the destination host by matching the first flow entry in the modified flow table. Then, the VLAN id is rewritten to 15, and the packet is looped back to the commodity switch, which, after doing a VLAN-to-port translation, forwards it to the destination host. Note that the only requirements for the commodity switch are manageability (to setup the VLANs on the access ports), support for 802.1Q (to do the VLAN un/tagging), and free trunk ports to be used for an uplink, which allows to use HARMLESS over basically any legacy Ethernet switch on the market today [47], [48]. One can observe that HARMLESS "consumes" the trunk ports for providing the Openflow capability, hindering their main purpose: providing uplink. To deal with the "loss" of trunks, in HARMLESS servers can have additional NICs to connect them directly (or via a spine switch) to provide inter-switch communication; see an example between *User A* and *User C*, and some additional basic flow rules in the modified flow tables in Fig. 2, respectively. The limitation due to the 12-bit length of the VLAN field affects the number of ports per commodity switch, not the total number of ports in the HARMLESS system. In particular, in HARMLESS VLAN tags are used to identify the physical ports of a commodity switch, the number of which is 2 orders of magnitude less than 4096 as current switches have an average number of 48 ports in one housing, requiring the same number of different VLAN tags. Correspondingly, the usable VLAN tags' domain is only bounded to one physical device and can be freely reassigned to another commodity switches' ports.

B. High Port Density at Low Cost

Below, we argue that HARMLESS strikes a fine balance between the cost-efficiency of COTS switches and the flexibility of softswitches in terms of deployment costs, data plane complexity, and performance.

HARMLESS Unifies the Advantages of Hardware and Software Switches: Thanks to the decoupling of raw forwarding functionality from OpenFlow, HARMLESS realizes an optimal separation of concerns, whereas the legacy hardware switch does just what it is best at, providing the port density and doing basic packet processing, i.e., VLAN manipulations to interact with HARMLESS, while the software switch component is again responsible for what it is ideal for, implementing the packet processing intelligence in a clear and extensible way. Observe that the softswitch does not need to match the port density of the commodity switch effectively removing the major cost component, NIC prices at the software switch.

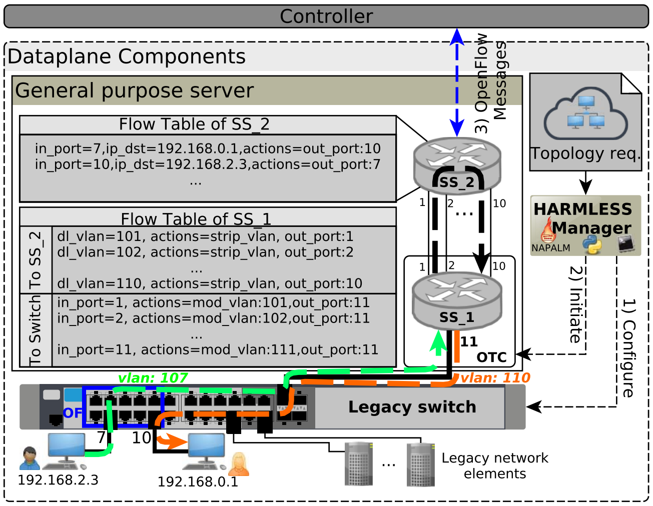
HARMLESS Leverages the Existing and Deployed Computing and Networking Infrastructure: The prerequisites of deploying HARMLESS is an inventory of commodity switches and a handful of spare bare-metal servers, readily available in many prospective SDN deployments like SOHO networks, small/medium-sized enterprises, private data centers, campus networks and private clouds. For these use cases, HARMLESS offers an instantaneous SDN transition path with *zero* initial expenditure, apart from the usual practice of server relocation, cabling, etc. Note, however, that the server running the OpenFlow component and the commodity switches do not even need to be co-located; in fact, the OpenFlow logic can be virtualized at a remote site or even delegated to a public cloud at the price of proper traffic forwarding and increased latency.

HARMLESS Is Cost-Efficient: Even in cases where spare servers or Ethernet switches are not available, purchasing them anew in a HARMLESS setup is still much less costly than purchasing equivalent SDN network gear from commercial suppliers. For the below CAPEX analysis, we assume that the legacy 48×1G+4×10G (or 48×10G+4×40G at 10G access) Ethernet switches are already on stock (if not, add another couple of hundred USD per switch), so only bare-metal servers for the OpenFlow components and 10G NICs (respectively, 40G) need to be purchased. We aggregate 12 access ports to each trunk port, over-committing the uplink at a similar rate as plain Ethernet networks [49], multiplexing up to 144 access ports (72 at 10G) to a single OpenFlow component. Accordingly, the CAPEX for a HARMLESS (HL for short) deployment with x ports are as follows:

$$\begin{aligned} C_{HL}^{1G} &= \$1400 \left\lceil \frac{x}{144} \right\rceil + \$400 \left\lceil \frac{x}{48} \right\rceil \\ &\quad + \$400 \left(\left\lceil \frac{x}{144} \right\rceil - \#extraCPU_{HL}^{1G} \right) \\ C_{HL}^{10G} &= \$1400 \left\lceil \frac{x}{72} \right\rceil + \$500 \left\lceil \frac{x}{24} \right\rceil \\ &\quad + \$400 \left(\left\lceil \frac{x}{72} \right\rceil - \#extraCPU_{HL}^{10G} \right) \\ \#extraCPU_{HL}^{1G} &= \begin{cases} 1, & \text{if } 0 < (x \bmod 144) < 97 \\ 0, & \text{otherwise} \end{cases} \\ \#extraCPU_{HL}^{10G} &= \begin{cases} 1, & \text{if } 0 < (x \bmod 72) < 49 \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

Similar to software switches, the first term accounts for the server, the second term for the NICs, and the third for additional CPUs. HARMLESS is the most cost-efficient SDN migration option, thanks to the high level of aggregation that reduces the number of costly servers and NICs as compared to pure softswitches, providing one order of magnitude more economical option (later, in Sec. IV we show how the CAPEX is affected when we deal with the "loss" of trunks).

HARMLESS Provides a Predictable and Arbitrarily Fine-Grained SDN Deployment Path: HARMLESS can seamlessly



```
in_port=7,ip_dst=192.168.0.1,actions=out_port:10
in_port=10,ip_dst=192.168.2.3,actions=out_port:7
...
```

```
dl_vlan=101, actions=strip_vlan, out_port:1
dl_vlan=102, actions=strip_vlan, out_port:2
...
dl_vlan=110, actions=strip_vlan, out_port:10
in_port=1, actions=mod_vlan:101,out_port:11
in_port=2, actions=mod_vlan:102,out_port:11
...
in_port=11, actions=mod_vlan:111,out_port:11
```

Fig. 3. Transparent HARMLESS: Software switch twice (S4).

convert wholesale legacy devices into SDN-enabled switches instantly, but it can also be configured to handle only a subset of the access ports or only a subset of the services/traffic, and thereby fits smoothly into any greenfield or incremental SDN deployment strategy. In a wider perspective, the association between ports and the softswitch that manages them is completely flexible in HARMLESS, that is, a single software switch instance can handle any combination of ports from any number of commodity switches (see Fig. 3 for an example). Not just that HARMLESS can be enabled in arbitrarily fine-grained steps, but it also allows to revert to any previous step in the migration process at no cost if a regression is experienced, by issuing a single management command to the commodity switch to turn off the specific VLAN tagging on access ports.

HARMLESS Eliminates Vendor Lock-in by Opening Up the Data Plane One Step Further: By leveraging commonplace Ethernet switches and open software switches running on commodity hardware, HARMLESS is vendor-neutral. In fact, HARMLESS carefully works around the fixed legacy switch OS and offloads forwarding intelligence to a dependable softswitch, this way freeing operators from the trap of vendor-specific OpenFlow extensions and proprietary management interfaces. Furthermore, HARMLESS enables both parts to evolve independently and distinct deployments could use different combinations of hardware/software processing.

HARMLESS Preserves Full Functionality: HARMLESS does not confine the forwarding functionality in any way and, as shall be shown, multiple real-world use cases can easily be realized. Since the whole forwarding logic is defined in OpenFlow and each packet is processed by the software switch, all OF programs are supported (e.g.,in case of multicast/broadcast messages, the softswitch can easily duplicate packets with different VLAN IDs). In case of a failure, furthermore, it can also be realized at the softswitch by noticing `PORT_DOWN` event for the physical port.

HARMLESS Provides Competent Performance: HARMLESS imposes extra load on the backplane fabric since, for supporting inter-port communication, traffic has to traverse the uplink port of the switch twice. Note that this extra load only manifests itself for intra-switch communication, but traffic between *different* HARMLESS switches managed by different HARMLESS servers induces no extra load at all since traffic goes through each element only once (see the orange dot-dashed line in Fig. 2). HARMLESS by default uses the same aggregation ratios as Ethernet (12 access ports per uplink), but one can easily leverage the flexibility of HARMLESS to avoid packet loss due to over-committing the uplinks, by reassigning access ports to uplinks so that the expected peek load never exceeds the uplink capacity and the load is evenly balanced across uplinks (we consider this later in Sec. IV).

HARMLESS Introduces Small Additional Latency: The additional softswitch in the loop entails increased latency irrespective to intra-, or inter-switch communication. In the next section, we show strong empirical evidence that latency introduced by HARMLESS is close to that of COTS and software-based SDN switches. We acknowledge, however, that the additional delay can be a drawback for delay-sensitive applications; obviously, HARMLESS was designed to support small-scale SDN deployments and it is not adequate for extremely high-performance and latency-critical workloads.

C. Transparent HARMLESS Architecture

Here we show that HARMLESS exposes a transparent view of the data plane to the controller. In the above strawman's setup the controller needs to tediously adapt the flow table for the offloaded forwarding setup by mapping ports to VLAN ids and vice versa. To avoid tailoring controller programs to the underlying HARMLESS layer, we introduce two transparent HARMLESS setups, where the idea is to install an *OpenFlow Translator Component (OTC)* between the controller and the software switch as an adaptation layer [50], [51].

In the *Proxy HARMLESS* , (consider an OTC translating the OpenFlow messages between the softswitch and the controller in Fig. 2), the OTC component acts as a proxy, disguising itself as a switch towards the controller and as a controller towards the softswitch, and rewrites all passing rules that match the input port (`in_port`) to match the corresponding VLAN id instead (`dl_vlan`), and output actions to appropriate VLAN rewriting rules (`out_port` → `mod_vlan`).

OTC, on the other hand, as depicted in Fig. 3 can be realized by an additional softswitch (denoted by SS_1), which is connected to the OF component configured by the controller (noted by SS_2) with as many patch ports as the number of managed access ports of the hardware appliance and dispatches packets to and from patch ports based on the VLAN ids (see Fig. 3). Therefore, this setup is called the *Software Switch Twice* (S4) configuration. There are two main advantages of this latter approach compared to the *Proxy* setup:

- as the software switch managed by the SDN controller has the same number of ports the hardware switch will use as OF-enabled ports, the control plane does not require any special attention to management capabilities, such as per port statistics, per port VXLAN assignment, etc.;
- it does not require any special application (e.g., proxy), just another instance of the same softswitch.

Due to these advantages, next we discuss S4 in detail.

In order to set up the S4 architecture, we developed *HARMLESS Manager* [52], an easy-to-use and freely available application built on top of Python and BASH.[4] HARMLESS Manager runs on the bare-metal server intended to materialize the software switch components and it automatically manages and queries the legacy Ethernet device itself via NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support[5]), the *de-facto* standard library in Python to interact with different router vendors' devices (e.g., Arista EOS, Juniper JunOS, Cisco IOS) via a unified API. Note, however, when a device is not supported by NAPALM, the simple requirements of HARMLESS can be easily done manually (by CLI/management interface).

Next, we discuss in detail how the HARMLESS Manager works. Consider again the example depicted in Fig. 3, where the hardware switch is intended to use only its first ten 1G ports (marked by the blue OF-labeled frame) and one 10G trunk port as an OpenFlow component leaving the rest of its ports to operate in the traditional way henceforward (denoted by the *Legacy network elements* connected to the switch). To this end, the network operator defines her needs in a simple configuration file (denoted by Topology req.) indicating the connection details to access the legacy device (e.g., management IP and port, username, password), the desired ports to use (e.g., Ethernet0, Ethernet10), and the details of the softswitch components (e.g., NIC port(s) for the incoming traffic, number of cores to isolate for packet processing, indicating the utilization of DPDK, controller details). Accordingly, HARMLESS Manager connects to the device and, in order to preserve the current functionality, it downloads and stores its running configuration, and automatically configures the device. In particular, it assigns different VLANs for the ten forwarding ports (in the example above, the VLANs are in range of $[101, 110]$) and, at the same time, configures the 10G uplink port to be used as a trunk port.

Let us summarize how the HARMLESS Manager sets up the software components. First it reads the configuration file parsing all parameters, e.g., the number of ports intended to be used on the hardware switch. Then the two desired software switches (SS_1 and SS_2) are instantiated and the physical port of the server connected to the hardware switch's trunk port is bound to one of them (SS_1). Next, according to the used number of ports on the hardware switch, the adequate number of logical port-pairs are added to the software switch instances, and get connected. Then the manager assembles and adds the corresponding flow rules to the switch that acts as the OTC (SS_1). In particular, the switch captures all VLAN tagged packets, removes the tags, and forwards them towards the other software switch (SS_2) via the logical links. The manager also assembles the flow rules for the packets coming back from the software switch managed by the controller application. When packets are received back through one of the logical ports, the corresponding VLAN tags are pushed onto them, and they are going to be sent out via the physical interface.

Once the two software switch instances are up and running, the controller will receive a connection request from an OpenFlow switch having 10 ports (materialized by SS_2). In case of the need to revert back to the previous operation, HARMLESS Manager can easily reset the hardware device using the dumped configuration.

These modifications render HARMLESS data plane transparent enabling to write controller programs ignoring the fact that the underlying data plane is realized with HARMLESS, to make controller programs portable between deployments, and to allow to invoke higher-level languages and policies to setup the data plane [53], [54].

IV. Evaluation

We evaluated practical aspects of HARMLESS and compared it against alternatives. We show the data plane performance in diverse use cases taken from practical networking applications and under different workloads, and we present the results side by side with the SDN migration cost analysis for each possible SDN switch option (softswitch, COTS and white-box switches, and HARMLESS). First, we describe our testbed and the measurement methodology, then we present the specific use cases, and finally we list the evaluation results.

A. Testbed and Methodology

Our testbed includes two IBM x3550 M5 servers with Intel Xeon E5-2620v3 processors and 64GB of memory running Debian Linux Jessie 8.0/kernel 4.9, each server equipped with an Intel X710 NIC (10G) and Intel XL710 (40G) NIC, respectively. The setup also contains two commodity switches, a `Cisco 3750X` (24×1G + 4×10G) and an `Arista 7048T` (48×1G + 4×10G), three COTS OpenFlow switches, an `HP 3500` (24×1G), an `Extreme X440` (28×1G + 2×10G), and a `Brocade ICX6610` (28×1G + 4×10G), and two white-box switches (`Quanta T1048` and `Edge-Core AS4610`) all supporting OpenFlow v1.3. The COTS switches represent the state-of-the-art in COTS SDN switching as of 2015, while the white-box switches are from the low-end market of 2016. The switches have the following flow table size limitations:

- `HP 3500`: 1 flow table in TCAM with max. 1500 rules, and 4 further logical tables (processed in software);
- `Extreme X440`: 1 flow table in TCAM with max. 255 flow rules, assuming each has limited length in terms of match fields, and another table for MAC and VLAN matching rules (also in TCAM); it does not receive flow mods in passive mode, i.e., without a controller.
- `Brocade ICX6610`: 1 flow table with max. 3000 rules in TCAM (half if rules match on both L2 and L3 headers), and no further tables.
- `Quanta T1048`: 1 flow table in TCAM with roughly 2000 flow rules and 6 logical tables for tens of thousands of flow rules and different layer matching (e.g., matching on both source and destination MAC address can only be implemented in a logical table).
- `Edge-Core AS4610`: supports multiple flow tables, one of them containing actions, carrying maximum only 3840 flow rules in TCAM (plus $24,576$ and $32,768$ flow rules for exact matching on destination MAC address and destination IP address, respectively).

In each experiment, one of the servers was configured to run NFPA [55], an Intel DPDK *pktgen*-based benchmarking tool, back-to-back with the system-under-test (SUT). For the software switch evaluations the SUT was provisioned on the other IBM server, running a stable version of OVS (v2.7.0)

[4] https://github.com/muuurk/HARMLESS

[5] https://github.com/napalm-automation/napalm

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

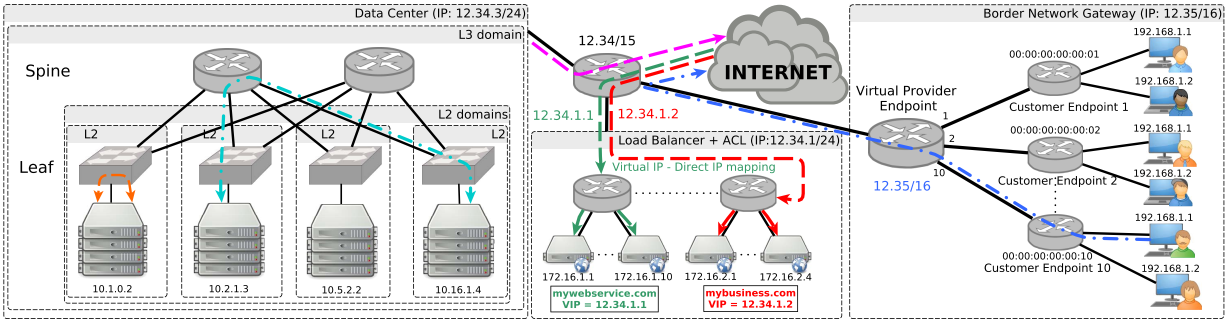


Fig. 4. Use cases in a service provider network.

and ESwitch[6] [24], both compiled with the recent stable Intel DPDK v16.11.1. The hardware switch option was evaluated on each of the COTS switches available in the testbed, while for HARMLESS the OpenFlow component was again configured on the IBM server running OVS (HARMLESS-OVS) or ESwitch (HARMLESS-ESwitch), connected to one of the commodity switches. Our HARMLESS setup implements the S4 setup discussed in Sec. III.

We used synthetic traffic traces, specially tailored to each use case (see below) to contain a configurable number of flows. The used packet trace was crafted in a way to simulate a 48-port switch setting by randomly tagging the packets with VLAN tags in the range of [1,48]. Note that packets were never dropped intentionally, instead the OpenFlow pipelines contain default catch-all rules to forward unmatched/dropped packets to the external port; our aim was to measure raw throughput and not whether the switches can filter traffic adequately (they can). With this configuration, packet loss only occurs when the SUT becomes a physical bottleneck and therefore the packet rate received at the packet generator is representative of the raw performance. Packets were minimum-sized (i.e., 64 bytes) and Receive Side Scaling (RSS) was turned on in multi-core setups [56], [57]. All measurements were conducted at 40G for at least 60 seconds [58]. At first the *packet rates were measured in a single-core setup*; note that the attainable throughput using a single core and PCIe x8 v3 bus speed is 15 Gbps (22 Mpps) with 64-byte packets; multi-core scalability is studied in a separate measurement round.

B. Use Cases

We considered 4 realistic use cases, from private data centers to telco gateways. All scenarios will be cast in a single hypothetical service provider's legacy network (see Fig. 4). The setup contains a smaller data center (DC) with 4 racks connected into a CLOS topology [59] with separate L2 domains at the leaves and an L3 domain as the spine [60], [61], an industrial-scale load-balancer [62], and a telco access gateway [63] that aggregates subscribers located behind Customer Endpoints (CEs) [64].

L2: The lower layer of the DC topology represents the *L2* use case, with each top-of-rack (ToR) switch provisioned as a separate *L2* domain; a sample L2 traffic flow is marked with orange in Fig. 4. While certain data centers may differ in the configuration of L2/L3 domains [65] this use case describes large L2 networks illustratively, to be migrated from traditional 802.1 to SDN with the aim of eliminating dependency on spanning trees and benefiting from centralized control [49], [66]–[68]. The OpenFlow pipeline consists of a single static MAC table containing a configurable number of entries; the synthetic traffic traces include as many flows (different MAC addresses) as the number of entries in the flow table.

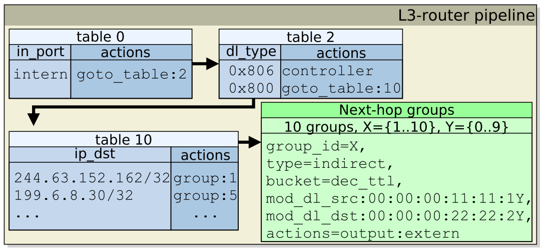

L3-router pipeline

table 0	
in_port	actions
intern	goto_table:2

table 2	
dl_type	actions
0x806	controller
0x800	goto_table:10

table 10	
ip_dst	actions
244.63.152.162/32	group:1
199.6.8.30/32	group:5
...	...

Next-hop groups
10 groups, X={1..10}, Y={0..9}

group_id=X,
type=indirect,
bucket=dec_ttl,
mod_dl_src:00:00:00:11:11:1Y,
mod_dl_dst:00:00:00:22:22:2Y,
actions=output:extern

Fig. 5. L3 pipeline.

L3: The *L3* use case embodies the upper layer of the CLOS network interconnecting the L2 islands, a common setup in DCs [60]; a sample traffic is marked with cyan in Fig. 4. In this setup the OpenFlow pipeline consists of multiple tables (see Fig. 5); after matching the input port in `table 0`, non-IP packets are sent to the controller (`table 2`) then in `table 10` IP lookup is performed with the corresponding actions diffused to 10 next-hop groups that do standard L3 packet processing. Again, different workloads were configured by setting different number of IP prefixes in `table 10` and a matching number of L3 flows in the synthetic traces.

Load Balancer and Access Control List: This use case captures the functionality of a web frontend, balancing incoming web traffic (TCP port 80) for different web services, each available at a unique IP address (see Fig. 4). The pipeline is given in Fig. 6; internal traffic is forwarded to the external port unconditionally, while ingress packets first take `table 1` that filters web traffic then `table 2` that distributes packets across backends based on the first bit of the source IP address. We set the number of web services to 100. Traffic traces were crafted so that 70% of packets go to a randomly chosen web service while the rest is filtered at the ACL.

Access Gateway: The telco *access gateway* consists of a Virtual Provider Endpoint (VPE) that serves Internet access to subscribers located behind CEs (see Fig. 4). For brevity, we identify CEs with the MAC address and we assume that the operator sets 10 CEs, each serving 20 users provisioned with a private IP address that is unique within the CE. The OpenFlow pipeline is given in Fig. 7. `Table 0` separates

[6] We thank L. Molnar *et al.* for providing us with the ESwitch source code as it is not open-source yet.

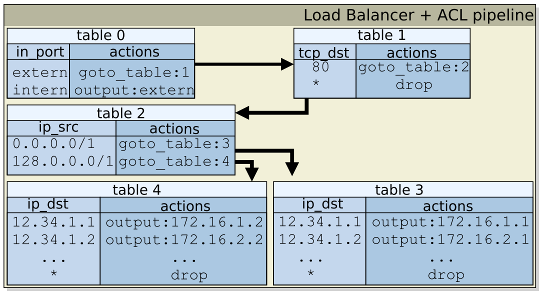


table 0

in_port	actions
extern	goto_table:1
intern	output:extern

table 1

tcp_dst	actions
80	goto_table:2
*	drop

table 2

ip_src	actions
0.0.0.0/1	goto_table:3
128.0.0.0/1	goto_table:4

table 4

ip_dst	actions
12.34.1.1	output:172.16.1.2
12.34.1.2	output:172.16.2.2
...	...
*	drop

table 3

ip_dst	actions
12.34.1.1	output:172.16.1.1
12.34.1.2	output:172.16.2.1
...	...
*	drop

Fig. 6. Load balancer and ACL pipeline.

user-to-network traffic on a per-CE basis from *network-to-user* traffic; *user-to-network* traffic goes to per-CE flow tables (`table 11`, etc.) that match the source IP address to identify the user and rewrite the source address to a unique public IP address (realizing a NAT), and finally to the IP routing table (`table 200`) that contains 10K random IP prefixes; packets from yet unseen users will in turn be forwarded to the controller to perform admission control and allocate a unique IP address. *Network-to-user* traffic, on the other hand, goes to `table 110` that matches on the destination IP to identify the user and the CE, swaps the destination IPs back to the private IP addresses, and then sends packets to the proper CE.

C. Measurement Results

Scalability and Standards-Compliance: Configuring the use cases on COTS and white-box switches proved far from trivial, due to the prohibitive flow table sizes and subtle restrictions on flow matching rules. The hardware switches support only a single flow table in TCAM and may or may not provide additional tables in software. Thus, multi-table OpenFlow pipelines had to be tediously collapsed into a single table by hand; in case of the white-box devices their software, e.g., Pica8 PicOS on `Quanta` and ONL+Indigo on `EdgeCore`, do this automatically. Unfortunately, even then the switches rapidly run out of TCAM space because of the flow-state explosion effects for which table collapsing is notorious [18]. In the *access gateway* use case for instance a separate flow entry must be created for every (user, CE, IP prefix) tuple, yielding so many entries that none of the hardware offerings could implement this use case. *Current COTS/white-box switches do not scale beyond small and medium workloads, and even in that case may require hand-tweaking the OpenFlow pipeline, while software switches and HARMLESS support even very large deployments*. Furthermore, one has to take into account the ramifications of the chip in each individual switch, e.g., the `HP` switch does not support static matching on MAC addresses, the `Brocade` switch does not support MAC rewrite, the `EdgeCore` switch cannot modify IP fields, only on slow-path.

Performance: Table II compares the raw packet rate measured in the *L2*, *L3*, *load balancer* and *access gateway* use cases with the hardware switches, the software switches, and the S4 configuration of HARMLESS. Recall that due to the attainable packet rate of a single CPU core the maximum accessible performance is up to 22 Mpps. Note that for each use case results are reported only for the hardware switches that could handle the use case. Our observations are as follows. First, as long as hardware OpenFlow switches manage to forward packets purely in the fast path they perform at wirespeed. However, as soon as a hardware switch runs out of TCAM space and forwarding falls back to the software slow path performance plummets. Note, however, that among the devices providing logical tables (`HP` and `Quanta`) only `HP` can use TCAM and logical tables simultaneously for all examined use cases. Furthermore, `Quanta` installs as many flow rules in its TCAM as it can ($2K$), and silently ignores the rest without notifying the controller. For instance, in the *L2* use case the `Extreme` switch could handle 100 flows at line rate ($1K$ and $2K$ flows with the `Brocade` and the `Quanta`, respectively) but could not tackle $1K$ flows at all ($10K$ for the `Brocade` and the `Quanta`). On the other hand, all hardware switches can support the relatively small flow table of the *load balancer* use case adequately (even though the `HP` proved to be slower).

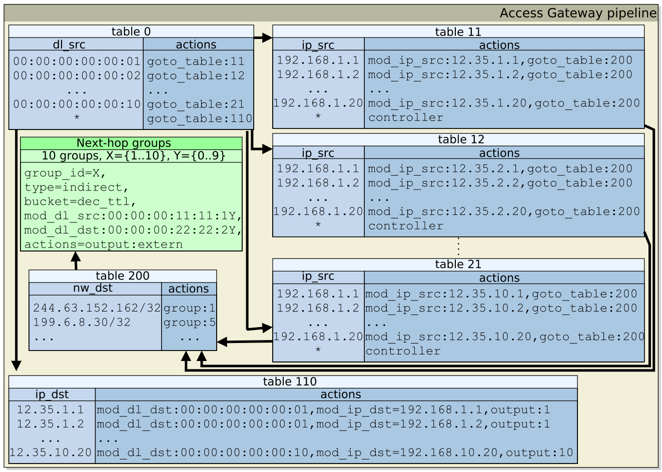


table 0

dl_src	actions
00:00:00:00:00:01	goto_table:11
00:00:00:00:00:02	goto_table:12
...	...
00:00:00:00:00:10	goto_table:21
*	goto_table:110

table 11

ip_src	actions
192.168.1.1	mod_ip_src:12.35.1.1,goto_table:200
192.168.1.2	mod_ip_src:12.35.1.2,goto_table:200
...	...
192.168.1.20	mod_ip_src:12.35.1.20,goto_table:200
*	controller

table 12

ip_src	actions
192.168.1.1	mod_ip_src:12.35.2.1,goto_table:200
192.168.1.2	mod_ip_src:12.35.2.2,goto_table:200
...	...
192.168.1.20	mod_ip_src:12.35.2.20,goto_table:200
*	controller

table 21

ip_src	actions
192.168.1.1	mod_ip_src:12.35.10.1,goto_table:200
192.168.1.2	mod_ip_src:12.35.10.2,goto_table:200
...	...
192.168.1.20	mod_ip_src:12.35.10.20,goto_table:200
*	controller

Next-hop groups
10 groups, X={1..10}, Y={0..9}

```
group_id=X,
type=indirect,
bucket=dec_ttl,
mod_dl_src:00:00:00:11:11:1Y,
mod_dl_dst:00:00:00:22:22:2Y,
actions=output:extern
```

table 200

nw_dst	actions
244.63.152.162/32	group:1
199.6.8.30/32	group:5
...	...

table 110

ip_dst	actions
12.35.1.1	mod_dl_dst:00:00:00:00:00:01,mod_ip_dst=192.168.1.1,output:1
12.35.1.2	mod_dl_dst:00:00:00:00:00:01,mod_ip_dst=192.168.1.2,output:1
...	...
12.35.10.20	mod_dl_dst:00:00:00:00:00:10,mod_ip_dst=192.168.10.20,output:10

Fig. 7. Access gateway pipeline.

Meanwhile the ESwitch-based OpenFlow softswitch performs close to line rate at small and medium sized workloads and only becomes worse at very large flow tables. Depending on the workload, *the HARMLESS-ESwitch combination attains a performance very close to that of the TCAM-based fast path of hardware switches and the best softswitches*, in the majority of the cases reaches up to 90–95% *and it robustly outperforms the hardware's slow-paths and the* `HP` *switch*.

Results with OVS are much worse, but then again the HARMLESS-OVS mix is very close to pure OVS. This suggests that *the performance of HARMLESS is eminently conditioned on the OpenFlow softswitch component*; here, the HARMLESS-ESwitch combination seems very appealing.

We measured throughput on multiple CPU cores (Fig. 9) under the larger workloads (namely, in the L3/100K and LB/10K use cases); this time, we use 128-byte packets as the Intel XL710 NIC cannot be saturated with smaller packets [69]. The results indicate that HARMLESS scales to multiple cores linearly, however the HARMLESS-ESwitch mix already achieves its maximum performance (which is much higher than OVS can attain with 6 cores) with only 4 cores.

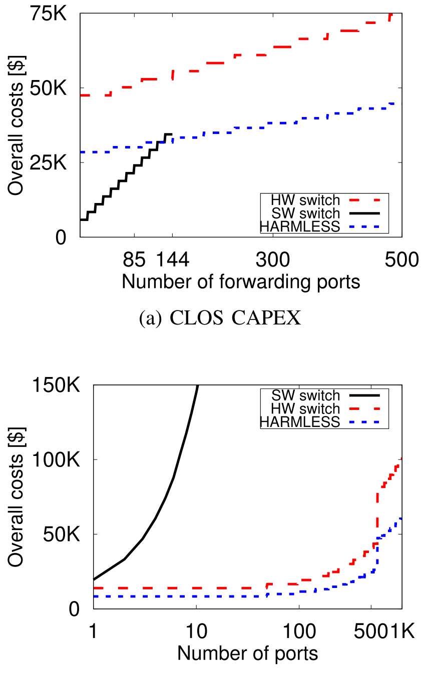
CAPEX: Fig. 8a compares the CAPEX of a greenfield deployment in the CLOS-based telco DC (the *L2* and *L3* use cases combined) as the function of access port density supported in ToR switches. Note that due to the different

TABLE II

THROUGHPUT MEASUREMENT RESULTS IN Mpps

		Brocade	**Extreme**	**Quanta**	**Edgecore**	**HP**	**ERFS**	**HL (S4) - ERFS**	**OVS**	**HL (S4) - OVS**
L2	**100**	22	22	22	22	0.0098	22	21.24	8.7	6.8
	1K	22	-	22	22	0.0096	21.9	21.18	6.7	5.5
	10K	-	-	-	22	0.0082	11.25	11	5.1	4.3
	100K	-	-	-	-	-	10.1	9.82	2.5	2.2
L3	**100**	-	22	22	22	2.938	20.1	18.13	6.0	5.6
	1K	-	-	22	22	2.938	18.11	15.7	5.7	4.6
	10K	-	-	-	22	0.757	12.86	12.3	4.6	3.8
	100K	-	-	-	-	-	11.12	10.65	2.3	2
LB	**100**	22	22	22	22	2.937	21.9	21.1	8.9	6.7
	1K	22	22	22	22	2.937	21.9	21.1	7.3	5.8
	10K	22	22	22	22	2.937	21.9	21.1	5.2	4.3
	100K	22	22	22	22	2.737	21.9	21.1	3.7	3.2
GW	**100**	-	-	-	-	-	16.1	11.83	7.3	5.7
	1K	-	-	-	-	-	12.62	11.12	6.7	5.7
	10K	-	-	-	-	-	12.77	11.06	3.8	4.8
	100K	-	-	-	-	-	13	11.05	2.3	2.1

Raw packet throughput in Mpps as the function of the workload size in the *L2*, the *L3*, the *load balancer* (LB) and the *access gateway* (GW) use cases.



(a) CLOS CAPEX

(b) GW CAPEX

Fig. 8. CAPEX at different deployment scales for (a) a full CLOS topology that integrates the *L2* and *L3* use cases, and (b) the *access gateway* (x in logscale). In the legend, SW: software switches, HW: hardware switches, HL: HARMLESS.

form factors the spine layer scales differently: purchasing four 48×10G hardware switches for the spine incurs a huge initial investment but can then scale to 48 leaf switches economically; in case of software switching, one leaf switch, offering similar aggregation ratios as a typical hardware device provides,[7] mounts 12×1G + 1×10G, thus we need *one server* with 12×10G capacity as the spine resulting in a small CLOS topology (providing 144 access ports at the most). Observe in Fig. 8a that in contrast to COTS devices and HARMLESS, where we are given 4 spine switches (the most expensive parts of the CLOS topology), the necessity of only one spine server is the reason why software switches involve less initial investment.

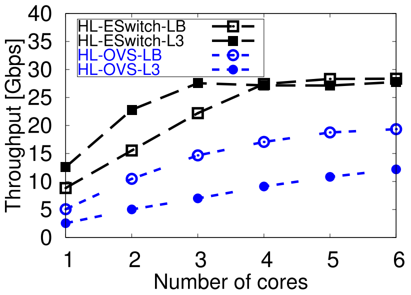


Fig. 9. HARMLESS Multi-core scalability: throughput (Gbps) with ESwitch and OVS (128-byte).

Since in case of HARMLESS the trunk ports of the legacy devices are used to provide the OpenFlow capability, special attention is needed in order to preserve the non-blocking 1:1 over-subscription ratio of the CLOS topology. Therefore, in HARMLESS a spine switch is comprised of a 48×10G+4×40G commodity switch plus a server with two 4×10G NICs for the softswitch component, and 2x100G NIC[8] with an additional CPU for compensating the inherent "loss" of uplink ports resulting in a sum of $3,900 per spine switch. On the other hand, a legacy leaf switch of 48×1G + 4×10G only requires a HARMLESS server with two 4×10G NICs resulting in an average price of $2,200 per leaf switch.

Fig. 8b gives the CAPEX for a telco *access gateway* greenfield SDN deployment in a simple tree topology (Fig. 4) with a depth of 3, consisting of 48×1G forwarding ports at the leafs, 48×10G aggregation switches in the middle, and one switching gear with 40G forwarding ports as the core (we considered the average price of $20,000 USD) offering an overall 1:1 over-subscription ratio. One can observe that when relying merely on software switches, expenses can easily reach high even for fewer number of ports. On the other hand, in case of the hardware devices and HARMLESS the steep cost steps arrive at 576 forwarding ports: those indicate the price of the 32×40G OpenFlow-enabled core switch, and the three 2×100G NICs for HARMLESS, respectively. Crucially, *in all cases HARMLESS is the most cost-efficient option, supporting roughly the same performance at the fraction of the price*: on average HARMLESS is 2–4 times less expensive than a softswitch-, a COTS-, or a white-box-based deployment, but

[7] Recall that a certain amount of over-subscription ratio is implied by the design of the hardware switches (e.g., 48×1G vs. 4×10G).

[8] We considered the average price of a Mellanox ConnectX-5 NIC of $1,300.

TABLE III

LATENCY OVER IN A BRIDGE AND IN THE L2/1K USE CASES WITH AND WITHOUT 10 Gbps BACKGROUND TRAFFIC

SDN switch	**Latency [μs]**			
	w/o load		w/ 10 Gbps load	
	Bridge	L2	Bridge	L2
SW-ESwitch	238±16	233±17	433±18	454 ±17
HL-ESwitch	268±19	265±19	471±21	491±19

TABLE IV

POWER CONSUMPTION AND RACK SPACE DEMAND

SDN switch	Power [W]			Rack
	Min	Max	Measured	
SW-ESwitch	70	230	162	3U
HW-HP	142	616	212	1U
HW-HP-SW			511	
HL-ESwitch	164	350	236	1.3U

the price difference can even reach to an order of magnitude. Our cost analysis assumes that legacy switches for HARMLESS are in stock; if not, HARMLESS is still 1.5–3 times cheaper due to the low price of commodity Ethernet switches; however, if Ethernet switches and spare servers are available in adequate number then, recall, HARMLESS incurs zero cost.

One might propose that since hardware switches can achieve higher throughput than their software-based counterparts, in the cost analysis their prices should be normalized correspondingly. However, the datasheets of the hardware appliances are neither comprehensive nor comparable: packet size is missing from the performance indicators, or the optimal packet sizes are different from switch to switch. In order to avoid any distortion in the results we did not consider this methodology.

Latency: In order to check whether the additional softswitch in the loop increases the latency of HARMLESS prohibitively, we conducted a series of latency measurements in various setups. The latency was measured by the Linux *ping* command and Table III gives the average and standard deviation results of 1000 measurements for a single port-forwarding rule in the OpenFlow pipeline and for the L2/1K workload. We have measured the delay with *no* (for baseline) and with *10 Gbps background traffic*. ESwitch's delay is around 230 μsec without background traffic reliably, with HARMLESS only 10% more thanks to that the underlying plain Ethernet switch is very fast (adding roughly 30–50 μsec to the softswitch latency). This performance difference also can be observed in the case of the 10 Gbps load in the background. VLAN untagging and tagging in the software switch does not induce apparent delay, and the computation overhead also seems negligible. The results indicate that *the additional softswitch does not introduce prohibitive latency in HARMLESS* , accordingly, it seems sufficient for anything but the most delay-critical applications. Note that by baseline we mean the latency of a pure software switch that could be used to realize the HARMLESS architecture. Note, since the use cases implied L2 and L3 synthetic traffic for measurements, the throughput is not impacted by the end-to-end/packet processing latency.

OPEX: Our cost analysis so far has accounted for the CAPEX component only, the operational costs were not considered at all even though OPEX can constitute a significant factor in the total spending. Table IV shows an evaluation of two important OPEX components. The energy consumption is estimated from the datasheets of the switches and the CPUs (note that the legacy Ethernet switch used in HARMLESS consumes less power than a full-scale SDN switch). The rack space occupancy is normalized for the standard 48×1G form factor: 1U for a hardware switch, 3U for the three 16×1G servers needed for a 48-port software switch, and for HARMLESS $1U$ for the legacy 48×1G switch and $1U$ for the 12×10G server, but the latter can handle 2 additional commodity switches as well which gives 1.3U normalized to 48 ports overall. Cabling might be more difficult though, since some high-speed uplinks that could otherwise be used for aggregation are allocated for HARMLESS; yet, the flexibility of access port assignment in HARMLESS may be exploited to optimize cabling costs. Note, that most of the cabling can remain the same: HARMLESS only requires to reuse the trunk ports (e.g., 4×10G) for the OF component, but at the same time, the HARMLESS server is accompanied with additional NICs (e.g., a NIC with 4×10G ports) to replace the otherwise lost trunk ports. Accordingly, only a little bit of rewiring is imposed by HARMLESS; the bulk of cabling that connects end devices to core switches does not need to be moved.

Overall, *the costs for operating HARMLESS are at the same level as that of the alternatives*.

V. RELATED WORK

We group the related work we cite around 4 topics.

SDN Control and Data Planes: Disruption-free updates are a key primitive to effectively operate SDN networks and maximize the benefits of their programmability. In [70] the authors study how to implement this primitive safely (with respect to forwarding correctness and policies), efficiently (in terms of consumed network resources) and robustly to unpredictable factors, such as delayed message delivery and processing. From the HARMLESS design perspective maximizing reliability and minimizing the latency induced by control plane actions are key, as the data plane forwarding is charged with an extra delay from the legacy hardware switch to the HARMLESS server. Striving towards the same goal, another major work, presented in [71], introduces a system that uncovers forwarding problems due to hardware or software failures in switches, by verifying that the data plane corresponds to the view that an SDN controller installs via the control plane. *The control plane of a HARMLESS network is exposed to the same dangers as ordinary software-defined networks. As the presented work focuses on the migration itself, making traditional networks into SDN ones, we regard these challenges to be orthogonal to our study.*

SDN Migration: The new levels of abstraction, programmability, and logically centralized control are important motivators for deploying SDN in operational networks [1], like enterprise networks [72], DC fabrics [68], transport networks [73], [74], WANs [5], [7], [75], and Internet exchanges [76]. Most of the deployments, however, involve the complete and irreversible overhaul of the existing legacy networking infrastructure. Incremental deployment strategies [15] seek to find a smoother migration path than a flag-day greenfield upgrade [1]–[4], [6]. Managing such a heterogeneous network however can become rather unwieldy due to the potential interference between coexisting legacy and SDN control planes; for instance, forwarding loops may be formed due to the legacy control plane masking certain forwarding decisions from the SDN controller [6]. *Due to its dataplane trans-*

parency, vendor-neutrality, and fine-grained upgradeability, HARMLESS easily fits into any of these SDN migration paths.

Hybrid SDN: Perhaps closest to HARMLESS is the hybrid SDN scheme Panopticon [3], which connects legacy device ports to SDN-capable switches using VLAN tagging similarly to HARMLESS. Yet, the objective of Panopticon is different: guaranteeing that each forwarding path traverses at least one SDN switch that can exert control over the traffic along that path. This requires the careful optimization of network policies for waypoint enforcement, which introduces path stretch; furthermore, Panopticon needs a nontrivial number of newly purchased SDN switches. *In contrast, HARMLESS is completely dataplane transparent being able to accommodate any SDN policy without special tweaking. Furthermore, HARMLESS can introduce the* existing *legacy network infrastructure under SDN control and hence is more economical.*

Fibbing is similar in this vein to HARMLESS [5], [74]: it endues a legacy network, employing a distributed routing protocol, with SDN control by the use of clever "lies", whereas certain network nodes are commanded by the controller to announce false routing information into the IGP. Fibbing, however, is bounded by the limitations of destination-based routing, while *HARMLESS opens up the full power of SDN to realize any forwarding and security policy in a transparent, verifiable, and debuggable framework.*

The authors' aim in LegacyFlow [77] is to provide a virtual OF datapath to the legacy appliances for translating only a set of OF actions into vendor specific configurations, this way creating a hybrid network. LegacyFlow requires more feature/capability from the HW device than HARMLESS and cannot support all OF operations (e.g., number of packets matched on a certain flow rule). *Furthermore, a virtual switch controller is limited to handle just one legacy device at a time.*

The way to adopt SDN is for each host to have a software switch, so the commodity hardware switch does not need to change: this is how companies install SDN in datacenters. HARMLESS offers an SDN adoption technique that requires to i) add a number of software switch instances to the network that is equal or less than the number of commodity switches present there, ii) connect the commodity switches' uplink ports with the servers that run those software switches, iii) set VLAN tags to the managed ports of the commodity switches. Installing software switches on each and every server, the number of which is generally at least an order of magnitude higher than that of the switches interconnecting them, might be a tedious work. Therefore we argue that HARMLESS is in fact the easier way for companies to adopt SDN.

The seminal work [60] on the softwarization of data center networks proposes to compute forwarding states in the controller and to deploy the corresponding flow entries into network hypervisors running on the data center servers interconnected with legacy network devices. In this so-called Network Virtualization Platform (NVP) the underlying commodity hardware switches do not need to change at all, instead the overlay connectivity required for the data paths among multi-tenant VMs is provided by virtual switches on the servers based on the tenants' configurations and the actual VM locations within the data center. The difference between HARMLESS and NVP are two-fold. First, the application domain we aim for is enterprise [78] and wide-area [79] networks instead of multi-tenant data centers, which completely changes the requirements from the ability of mapping virtual data paths on the physical network to the ability of controlling the network itself. Second, by moving the goal so, *in HARMLESS the implementation of the logical network does not necessitate to install a software switch on each and every server, only the network devices have to be softwarized, which is done by at most one software switch per commodity hardware switch.*

Hard vs. Softswitches: Lately, there has been significant research in hardware switch architectures (RMT [80], packet transactions [31], P4 [81], TCAM caches [25]) and software switch design (multi-threaded switches [21], [22], OVS [18], ESwitch [24], PISCES [23]), but the debate still seems far from concluded [27]–[30]. Bridging the gap between the two extremes has been a recurring theme for long [26], e.g., recent trends for NIC offloading [29], `switchdev` [82], similar ideas in hybridization for load balancing [62]. *Nevertheless, we see HARMLESS rather as an architectural leap than another hybrid hardware-software design; nothing in HARMLESS enforces that the commodity switch be a hardware device and the OpenFlow component be a softswitch, it is just that this setup realizes a particularly cost-effective combination.*

VI. Conclusion

SDN has grown out of the "niche status" and found important use in communication networks. However, there still exist areas it has not penetrated, mainly service provider networks and smaller businesses with less technically savvy IT staff. In this paper we presented HARMLESS, a new SDN design to offer an attractive deployment path for such cases.

The main idea in HARMLESS is opening traditional black-box network gear and virtualizing the switch OS in a separate softswitch component. HARMLESS allows an operator to start experimenting with SDN instantaneously: by connecting the trunk port of a legacy Ethernet switch to a spare x86 server an operator can immediately engage with OpenFlow controller programs with zero initial investment. Later, any combination of legacy ports and switches can be connected to the HARMLESS software switch to incrementally reach a full SDN deployment. Thanks to transparency, the developed controller programs are portable between deployments.

HARMLESS realizes an appealing combination of hardware and software switching, with the hard switch providing the port density and the softswitch delivering programmability. Our comprehensive CAPEX analyses on realistic SDN migration scenarios indicate that HARMLESS attains the most economic SDN migration strategy today, with performance close (90–95%) to, and in some cases even higher than, that of the alternatives. HARMLESS is exempt from the dataplane quirks and performance regressions experienced with COTS OpenFlow appliances. With the continuous evolution of software switching and general-purpose packet processing solutions, achievable throughput with HARMLESS will further improve.

References

[1] A. Bhalgat *et al.*, "SDN migration considerations and use cases," ONF Solution Brief, Open Netw. Found., Palo Alto, CA, USA, Tech. Rep. ONF TR-506, 2014.

[2] M. McNickle, *With Hybrid SDN Deployment, no Need for Network Forklift*. Newton, MA, USA: Tech Target Search SDN, 2013.

[3] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *Proc. ATC*, 2014, pp. 333–345.

[4] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, "One step at a time: Optimizing sdn upgrades in isp networks," in *Proc. INFOCOM*, May 2017, pp. 1–9.

[5] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," in *Proc. SIGCOMM*, Aug. 2015, pp. 43–56.

[6] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 70–75, Apr. 2014.

[7] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[8] A. Gonsalves, *Microsoft SDN Stack to Challenge Cisco, VMware*. Newton, MA, USA: Techtarget, 2016.

[9] C. Ma. (2014). *SDN Secrets of Amazon and Google*. [Online]. Available: https://bit.ly/2OPkpZi

[10] S. Ong, "Migrating to SDN: Planning for a smooth transition," Brocade, San José, CA, USA, White Paper, Jul. 2014.

[11] N. Computing. (2017). *SDN: Time to Move On, Gartner Says*. Gartner Report. [Online]. Available: https://ubm.io/2LmrLBu

[12] A. Lerner. (2014). *The State of SDN Adoption*. Gartner Blog Network. [Online]. Available: https://gtnr.it/33NlCVo

[13] R. Chua. (2018). *State of SDN and NFV Hype or Reality*. SDxCentral. [Online]. Available: https://bit.ly/2OOy4jn

[14] L. Csikor and D. P. Pezaros, "End-host driven troubleshooting architecture for software-defined networking," in *Proc. GLOBECOM*, Dec. 12017, pp. 1–7.

[15] M. K. Mukerjee *et al.*, "Understanding tradeoffs in incremental deployment of new network architectures," in *Proc. CoNEXT*, 2013, pp. 271–282.

[16] I. Pepelnjak. (2016). *Q&A: Vendor OpenFlow Limitations*. [Online]. Available: https://bit.ly/2RmBs6G

[17] I. Pepelnjak. (2016). *Table Sizes in OpenFlow Switches*. [Online]. Available: https://bit.ly/385gGhV

[18] B. Pfaff *et al.*, "The design and implementation of Open vSwitch," in *Proc. NSDI*, 2015, pp. 117–130.

[19] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about SDN flow tables," in *Proc. PAM*, 2015, pp. 347–359.

[20] *Guide: Data Plane Development Kit for Linux*, Intel, Santa Clara, CA, USA, 2015.

[21] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Towards high performance virtual routers on commodity hardware," in *Proc. CoNEXT*, 2008, p. 20.

[22] M. Dobrescu *et al.*, "RouteBricks: Exploiting parallelism to scale software routers," in *Proc. SOSP*, 2009, pp. 15–28.

[23] M. Shahbaz *et al.*, "PISCES: A programmable, protocol-independent software switch," in *Proc. SIGCOMM*, 2016, pp. 525–538.

[24] L. Molnár *et al.*, "Dataplane specialization for high-performance openflow software switching," in *Proc. SIGCOMM*, 2016, pp. 539–552.

[25] M. Casado, T. Koponen, D. Moon, and S. Shenker, "Rethinking packet forwarding hardware," in *Proc. HotNets*, 2008, pp. 1–6.

[26] D. Moon *et al.*, *Bridging the Software/Hardware Forwarding Divide*. Berkeley, CA, USA: Univ. California, Berkeley, 2010.

[27] K. Argyraki *et al.*, "Can software routers scale," in *Proc. PRESTO*, 2008, pp. 1–6.

[28] G. Pongrácz, L. Molnár, Z. L. Kis, and Z. Turányi, "Cheap silicon: A myth or reality? Picking the right data plane hardware for software defined networking," in *Proc. HotSDN*, 2013, pp. 103–108.

[29] J. Gross, A. Lambeth, B. Pfaff, and M. Casado, "The rise of soft switching,—Part I, II, III," *Netw. Heresy*, 2011. [Online]. Available: https://networkheresy.wordpress.com/

[30] G. Ferro, *Soft Switching Fails at Scale*. EtherealMind, 2011. [Online]. Available: https://etherealmind.com/soft-switching-fails-at-scale/

[31] A. Sivaraman *et al.*, "Packet transactions: High-level programming for line-rate switches," in *Proc. SIGCOMM*, 2016, pp. 15–28.

[32] K. Kogan, S. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "SAX-PAC (scalable and expressive packet classification)," in *Proc. SIGCOMM CCR*, Aug. 2014, vol. 44, no. 4, pp. 15–26.

[33] L. Csikor *et al.*, "Tuple space explosion: A denial-of-service attack against a software packet classifier," in *Proc. CoNEXT*, 2019, pp. 292–304.

[34] L. Csikor, C. Rothenberg, D. P. Pezaros, S. Schmid, L. Toka, and G. Rétvári, "Policy injection: A cloud dataplane DoS attack," in *Proc. SIGCOMM (Demo)*, 2019, pp. 147–149.

[35] J. Maddison. (2016). *Why Networks Need ASICs*. Fortinet Blog. [Online]. Available: https://ubm.io/33RRyrw

[36] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A SOFT way for openflow switch interoperability testing," in *Proc. CoNEXT*, 2012, pp. 265–276.

[37] NoviFlow. (2019). *NOVISWITCH—Swithing Made Smarter*. [Online]. Available: https://noviflow.com/noviswitch/

[38] CORSA. (2019). *SDN Done Right. For the Physical Network*. [Online]. Available: https://www.corsa.com/

[39] "The world's fastest & most programmable networks," Barefoot Netw., Santa Clara, CA, USA, White Paper, 2016. [Online]. Available: https://www.barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/

[40] A. Pavlidis *et al.*, *Overview of SDN Pilots Description and Findings: Part A*, document Deliverable D7.1, 2017.

[41] N. Gaur, *Fundamentals of VLANs: Router on a Stick*. Sydney, NSW, Australia: CCENT/CCNA R&S Study Group, 2014.

[42] *Network Address Translation on a Stick*. document 6505, Tech. Study, Cisco, 2008.

[43] A. Lunn, V. Didelot, and F. Fainelli, "Distributed switch architecture," in *Proc. Netdev 2.1, Tech. Conf. Linux Netw.*, 2017, pp. 1–7. [Online]. Available: https://netdevconf.info/2.1/session.html?lunn_didelot_fainelli

[44] P. Congdon. (2008). *Virtual Ethernet Port Aggregator—Standards Body Discussion*. [Online]. Available: https://bit.ly/2Pg6JFx

[45] J. Pelissier. (2013). *VNTag 101*. [Online]. Available: https://bit.ly/3611Zuh

[46] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks, Amendment 4: Provider Bridges*. Standard 802.1ad-2005, 2005.

[47] *Campus Network for High Availability Design Guide*, Design Guide, Cisco, San Jose, CA, USA, 2008.

[48] *Campus Networks Reference Architecture*, Juniper, Sunnyvale, CA, USA, 2010.

[49] H. Hudson, *Extending Access to the Digital Economy to Rural and Developing Regions*. Cambridge, MA, USA: MIT Press, 2002.

[50] M. Kuzniar, P. Peresini, and D. Kostić, "Providing reliable FIB update acknowledgments in SDN," in *CoNEXT*, 2014, pp. 415–422.

[51] P. Perešíni, M. Kuzniar, and D. Kostic, "Rule-level data plane monitoring with monocle," in *Proc. SIGCOMM CCR*, 2015, vol. 45, no. 5, pp. 595–596.

[52] M. Szalay, L. Toka, G. Rétvári, G. Pongrácz, L. Csikor, and D. P. Pezaros, "HARMLESS: Cost-effective transitioning to SDN," in *Proc. SIGCOMM Posters Demos*, 2017, pp. 91–93.

[53] N. Foster *et al.*, "Frenetic: A network programming language," in *Proc. ICFP*, Sep. 2011, pp. 279–291.

[54] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with pyretic," *USENIX Mag.*, vol. 38, no. 5, pp. 40–47, 2013.

[55] L. Csikor, M. Szalay, B. Sonkoly, and L. Toka, "NFPA: Network function performance analyzer," in *Proc. NFV-SDN*, Nov. 2015, pp. 15–17.

[56] T. Herbert. (2016). *Scaling in the Linux Networking Stack*. Linux Documentation. [Online]. Available: https://goo.gl/MRldGX

[57] Microsoft. (2016). *MSDN: Introduction to Receive-Side Scaling*. [Online]. Available: http://goo.gl/BpoErm

[58] S. Bradner and J. McQuaid, *Benchmarking Methodology for Network Interconnect Devices*, document RFC 2544, 1999.

[59] M. Al-Fares, M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. SIGCOMM*, 2008, pp. 63–74.

[60] T. Koponen *et al.*, "Network virtualization in multi-tenant datacenters," in *Proc. NSDI*, 2014, pp. 203–216.

[61] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. IMC*, 2010, pp. 267–280.

[62] R. Gandhi *et al.*, "Duet: Cloud scale load balancing with hardware and software," in *Proc. SIGCOMM*, 2014, pp. 27–38.

[63] Intel. *Network Function Virtualization: Virtualized Bras With Linux and Intel Architecture*. Accessed: Dec. 20, 2019. [Online]. Available: https://docplayer.net/5271505-Network-function-virtualization-virtualized-bras-with-linux-and-intel-architecture.html

[64] S. K. N. Rao, "SDN and its USE-CASES-NV and NFV," NEC Technol. India Ltd., New Delhi, India, White Paper NEAD-WP-001, 2014.

[65] Cisco. (2011). *Cisco Data Center Infrastructure 2.5 Design Guide*. [Online]. Available: https://goo.gl/kW78VM

[66] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: A scalable Ethernet architecture for large enterprises," in *Proc. SIGCOMM*, 2008, pp. 3–14.

[67] S. Halabi, *Metro Ethernet*. Indianapolis, IN, USA: Cisco Press, 2003.

[68] R. N. Mysore *et al.*, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, 2009.

[69] *Intel Ethernet Converged Network Adapters XL710 10/40 GbE*, Intel Corporation, Datasheet, Santa Clara, CA, USA, 2015.

[70] S. Vissicchio and L. Cittadini, "Safe, efficient, and robust SDN updates by combining rule replacements and additions," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 3102–3115, Oct. 2017.

[71] P. Peresini, M. Kuzniar, and D. Kostic, "Dynamic, fine-grained data plane monitoring with monocle," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 534–547, Feb. 2018.

[72] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise WLANS with Odin," in *Proc. HotSDN*, 2012, pp. 115–120.

[73] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the 'one big switch' abstraction in software-defined networks," in *Proc. CoNEXT*, 2013, pp. 13–24.

[74] M. Chiesa, G. Rétvári, and M. Schapira, "Lying your way to better traffic engineering," in *Proc. CoNEXT*, 2016, pp. 391–398.

[75] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. SIGCOMM*, 2013, pp. 15–26.

[76] A. Gupta *et al.*, "SDX: A software defined Internet Exchange," in *Proc. SIGCOMM*, 2014, pp. 551–562.

[77] F. Farias, I. Carvalho, E. Cerqueira, A. Abelém, C. E. Rothenberg, and M. Stanton, "Legacyflow: Bringing openflow to legacy network environments," *OFELIA Summer School*, to be published.

[78] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Proc. SIGCOMM*, 2007, pp. 1–2.

[79] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *Proc. SIGCOMM*, 2013, pp. 3–14.

[80] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proc. SIGCOMM*, 2013, pp. 99–110.

[81] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[82] S. Feldman, "Rocker: Switchdev prototyping vehicle," in *Proc. Netdev*, 2015, pp. 1–17.

Levente Csikor received the M.Sc. and Ph.D. degrees from the Budapest University of Technology and Economics, in 2010 and 2015, respectively. He is currently a Senior Research Fellow with the National University of Singapore. Before joining NUS, he was a Research Associate with INTRIG, University of Campinas, in 2018, and with the School of Computing Science, University of Glasgow, in 2017. His interests include data plane performance of different software-based network functions, network programmability, and denial-of-service attacks.

Márk Szalay is currently pursuing the Ph.D. degree with the HSNLab, Budapest University of Technology and Economics. His main research interests include hardware (router/switch/NIC) design, network programming, software-defined networking, and network function virtualization.

Gábor Rétvári received the M.Sc. and Ph.D. degrees in electrical engineering from the Budapest University of Technology and Economics in 1999 and 2007, respectively. He is currently a Senior Research Fellow with the HSNLab and a Visiting Professor with the Ericsson Research, Hungary. He maintains several open source scientific tools written in Perl, C, and Haskell. His research interests include all aspects of network routing and switching, the programmable data plane, and the networking applications of computational geometry and information theory.

Gergely Pongrácz graduated from the Budapest University of Technology and Economics in 2000. In 2004, he became a Research Engineer at Ericsson. He is currently an Expert with the Ericsson Research in the area of programmable dataplane. He is working on NFV and SDN topics, especially in the programmable networking area. His projects resulted in well received articles and demos, such as an article at ACM SIGCOMM in 2016 or demos at the Mobile World Congress in 2015 and 2017.

Dimitrios P. Pezaros (S'00–M'04–SM'14) received the B.Sc. and Ph.D. degrees in computer science from Lancaster University. He is currently a Professor of computer networks and the Founding Director of the Networked Systems Research Laboratory (netlab), School of Computing Science, University of Glasgow. He has published widely in the areas of computer communications, network and service management, and resilience of future networked infrastructures, and has received significant funding for his research in the above areas from public funding agencies and the industry. He is a Senior Member of the ACM. He is a Chartered Engineer.

László Toka received the Ph.D. degree from Telecom ParisTech in 2011. He worked at Ericsson Research from 2011 to 2014. Then, he joined the academia with research focus on software-defined networking, cloud computing, and artificial intelligence. He is currently an Assistant Professor with the Budapest University of Technology and Economics, the Vice-Head of the HSNLab, and a member of both the MTA-BME Network Softwarization and the MTA-BME Information Systems Research Groups.