The Price for Programmability in the Software Data Plane: The Vendor Perspective

Tamás Lévai, Gergely Pongrácz, Péter Megyesi, Péter Vörös, Sándor Laki, Felicián Németh, and Gábor Rétvári, *Member, IEEE*

***Abstract*— The killer features of the next-generation 5G mobile standard, including mobile edge computing and network slicing, will be very difficult to support with traditional fixed-function network appliances. Rather, the 5G core will depend on programmable switches, which allow packet processing functionality to be reconfigured on the fly in order to deploy virtualized network functions and service chains instantaneously. With 5G on the close horizon, it has become crucial to identify the price for programmability in the software data plane, considering the expected complexity and scale of the next-generation mobile core. In this paper, we report on a multi-year data-plane scalability study we have conducted for a large mobile vendor. Our results paint a rather pessimistic picture on the current landscape of the programmable software data plane. We find that the prominent programmable switches either do not provide all the features necessary to implement 5G telco pipelines efficiently or struggle to meet the scale, and the performance operators have come to expect from conventional fixed-function appliances. The only exception, ESwitch, remains proprietary. We call for further work on data-plane scalability and sketch some directions for future research.**

***Index Terms*— 5G, software-defined networks, programmable data plane, software switch, scalability, universal scalability law.**

I. Introduction

MAJOR telcos and operators are working hard to finalize 5G, the upcoming fifth generation mobile wireless standard. Slated to become commercially available by 2020, 5G will bring orders of magnitude improvement in system capacity, access speed, latency, and energy efficiency, in order to support a broad variety of vertical industries. 5G introduces a completely new service architecture, including mobile edge computing, which provides cloud-computing capabilities in close proximity to subscribers in order to accelerate access to content, and network slicing, a virtualization mechanism to dynamically initiate service overlays, improving the typical network operator innovation cycle and time-to-market [1], [2].

The current mobile core architecture, provisioned using appliance-based (black-box) network gear, fixed-function middleboxes, and proprietary hardware, no longer suites the needs of 5G [3]–[5]. Consequently, *network function virtualization* (NFV), which allows instantaneous deployment of different network functions, such as firewalls or encryption, on virtual machines (VMs) or white-box network gear, and *software-defined networking* (SDN), permitting to re-program the data plane on the fly to organize VNFs into arbitrary service chains, are considered key architectural enablers to realize 5G [4], [5]. NFV and SDN can address deployment barriers by reducing equipment costs (CAPEX) and operational expenditures (OPEX) and may also alleviate rising scalability concerns, thanks to the "limitless" elasticity of cloud-like deployments.

A fundamental building block of the SDN/NFV stack is the *programmable software switch*, a software-based packet processing pipeline (with possible hardware-acceleration). A programmable switch can manipulate flows based on essentially any combination of L2/L3/L4 header fields and allows the forwarding functionality to be dynamically reconfigured via an open and standardized control-plane interface [6], [7]. In the simple case the software switch merely passes packets between isolated VMs that run VNFs [8]–[13], whereas in integrated pipelines the switch may also run some or all the VNFs embedded into the data plane [14]–[16].

With the available data rates, service diversity, and the sheer scale, 5G is going to push the limits of what is achievable with the current programmable data plane [9]–[17]. Consequently, vendors are facing the question which programmable switch to base the next 5G product line onto, whether the chosen technology will support the massive 5G workloads, and whether it will facilitate economic growth of the product line into the future. The main goal of this paper is *to summarize the experiences of a major telco vendor* gained in the scalability analysis of current programmable data-plane technologies and *to evaluate the price 5G operators will pay, in terms of performance, latency, CAPEX/OPEX, and energy consumption, for the improved flexibility provided by programmable switches* compared to fixed-function appliances. The present paper fully takes the stance of a telco vendor: we focus more on presenting the results and less on analyzing the reasons behind the scalability traits particular switches exhibit.

Manuscript received March 14, 2018; revised July 20, 2018; accepted August 20, 2018. Date of publication September 19, 2018; date of current version December 11, 2018. *(Corresponding author: Tamás Lévai.)*

T. Lévai and F. Németh are with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, H-1111 Budapest, Hungary (e-mail: levai@tmit.bme.hu; nemethf@tmit.bme.hu).

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

P. Megyesi is with LeanNet, H-1174 Budapest, Hungary (e-mail: megyesi@leannet.eu).

P. Vörös and S. Laki are with the Faculty of Informatics, Eötvös Loránd University, H-1117 Budapest, Hungary (e-mail: vpetya@mensa.hu; lakis@elte.hu).

G. Rétvári is with the MTA-BME Information Systems Research Group, Budapest University of Technology and Economics, H-1111 Budapest, Hungary (e-mail: retvari@tmit.bme.hu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2018.2871307

The contributions of the paper are five-fold.

A taxonomy for data-plane scalability. We give an overview on the critical aspects concerning data-plane scalability. We describe the most important problem dimensions, we identify crucial performance descriptors, and we also point out some common traps. This is the main topic in Section II.

TIPSY, a data-plane scalability benchmarking tool. Contrary to earlier tools and benchmarks [18]–[28], TIPSY was designed from the ground up for the black-box scalability analysis of networked software: it allows to fire up a standard telco pipeline and set scaling parameters in only a couple of easy configuration steps, supports multiple scaling modes and pluggable backends, permits turn-key measurements, and provides automated visualization. TIPSY is described in Section III.

A suite of standard 5g telco pipelines. We define 10 standard telco pipelines, identifying the typical 5G VNFs as standalone micro-benchmarks and then synthetizing these basic building blocks into meaningful macro-benchmarks to allow experimentation at scale. The pipelines are described in Section IV.

A scalability study for the programmable software data plane. We provide the performance baseline for 8 prominent programmable switches and then, concentrating on the most efficient four products, we present a detailed scalability analysis on complex 5G pipelines. See Section V.

Scalability is of concern in software switches. Our analysis reveals that scalability, considering the expected size and load of the future 5G mobile core, remains a challenge for most programmable software switches. Consequently, deploying the programmable data plane would require unreasonable capital and operational expenditures compared to traditional black-box designs. Our experiences are summarized in Section VI and we conclude the paper in Section VII.

II. DATA-PLANE SCALABILITY: A TAXONOMY

Operating an SDN/NFV stack at the 5G scale requires the underlying software switches to handle enormous *static workload*, tracking packets through a hierarchy of match-action tables, each containing possibly thousands of entries, at a rate of millions of packets per second. Moreover, switches must also support a massive *dynamic workload*, since the match-action pipeline may be modified multiple thousand times per second [2]. The main goal in a scalability analysis is to decide whether a particular switch tolerates growing static and dynamic load economically and efficiently.

A scalability benchmark takes as *input* a critical system parameter, like the size of the workload or the number of CPU threads participating in packet processing, and the *output* is one or more performance metrics (packet rate, latency, jitter) characterizing the efficiency of the system along the input trajectory. The final result is a benchmark report, e.g., a chart with the input parameter on the x axis and the measured change in the output metric on the y axis.

On the traces of [29], we distinguish three fundamental "metric of size" descriptors, or *scalability dimensions*, which can be scaled separately or jointly in order to obtain scalability characterizations in different operational domains (see Fig. 1).

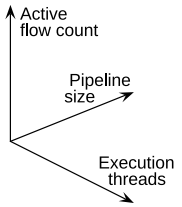


Fig. 1. Scalability dimensions.

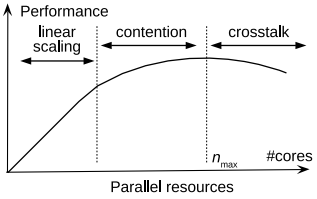


Fig. 2. Universal scalability law.

- *Concurrency.* The amount of computing resources, in terms of the number of CPU cores, dedicated to parallel packet processing. Of course, the more cores a switch may use the higher the throughput.
- *Pipeline size.* The *spatial complexity* (e.g., the number of VNFs and service chains) and the *temporal complexity* (i.e., the rate and type of updates applied during run-time) of the workload running on a switch. Higher pipeline complexity usually induces worsening performance.
- *Active flow count.* The number of individual transport protocol sessions injected into the switch, which typically represent the fundamental units of unsplittable traffic that must be scheduled to the same CPU thread (see later).

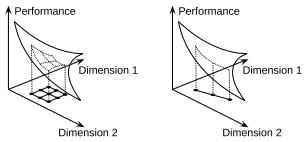
The output is a performance descriptor that will represent the "goodness" of the switch at a particular scale, like packet rate and transmission speed, latency and jitter, drop rate, etc. Sometimes, these quantities are interpreted side-by-side, e.g., the maximum packet rate achievable with at most 10^{-4} drop probability (an "RFC2544-style" measurement [30]).

A. Modeling Scalability

The Universal Scalability Law (USL, [29], [31]) is often used to reason about benchmark results in the CPU-resources input domain. This law characterizes the maximum parallel execution gain $X(n)$ achievable with a system running on n CPU cores, as the function of the *contention* σ, the fraction of serialized (non-parallelizable) portion of the work, and the *crosstalk* κ, which may happen due to the interference between each pair of worker threads in the system:

$$X(n) = \frac{\lambda n}{1 + \sigma(n-1) + \kappa n(n-1)}. \tag{1}$$

For $\sigma > 0$ and $\kappa = 0$ the parallel speedup asymptotically approaches the scalability limit $1/\sigma$ (Amdahl's Law), while for $\kappa > 0$ we get a speedup peak at $n_{\max} = \sqrt{\frac{1-\sigma}{\kappa}}$ and retrograde scalability trend afterwards (see Fig. 2). Note that TIPSY, our benchmarking system, supports automatic fitting of the USL for multi-core measurement results.



(a) outer (b) joint

Fig. 3. Scaling modes: outer and joint scaling.

B. Common Pitfalls

Software switches typically run in polling mode, whereby working threads are bound to a port's receive packet queue permanently and drain the queue in tight loop to process packet batches in a run-to-completion manner. The NIC will, in turn, dispatch packets to CPU cores by the IP 5-tuple (RSS). It is important to understand how this architecture interacts with each scalability dimension to obtain correct benchmarks.

1) Scaling Modes: In order to visualize multi-dimensional scalability benchmarks usually some forms of scaling-dimension reduction is used. For instance, reducing the number of relevant input dimensions to 2 and leaving the rest unchanged produces a two-dimensional plot for the *outer product* of the input regimes. Conversely, growing multiple input parameters simultaneously will yield a single *joint* input dimension on the x axis. TIPSY supports both the "joint" and the "outer" scaling modes out of the box (see Fig. 3).

2) Interactions Between Scalability Dimensions: Although representing orthogonal operational domains, scalability dimensions are not independent. For instance, measuring with a single active flow injected into a software switch will not result correct multi-core scalability readings, since the single active flow will always be scheduled by the NIC to the same CPU core. To obtain correct results one must increase the active flow count, the pipeline size (to handle the additional flows) and the CPU core count *jointly*, taking care to fix the scaling unit for the single-core baseline and keeping the transaction count per worker thread constant afterwards. TIPSY was designed to easily support such complex scalability benchmarks.

3) Pipeline and Platform Tuning: The performance of a programmable switch drastically depends on certain minute implementation and platform details [10], [18]–[26], [32]. It is our belief, however, that a vendor, and especially an operator, should not need to master a data-plane technology to extract reasonable performance; correspondingly we deliberately avoided fine-grained backend-specific tuning. Rather, we created "best-intention", but admittedly naive, implementations for the telco pipelines and we made some minimal effort to configure the System-Under-Test (SUT) for best overall performance; e.g., all backends use the Intel DPDK user space networking kit running on isolated CPUs for maximum throughput [26], [33]. TIPSY, thanks to the modular architecture, allows to easily swap inefficient implementations for more efficient ones or do comparison studies in the future.

III. TIPSY: AN AUTOMATED BENCHMARK TOOL

TIPSY, the *Telco pIPeline benchmarking SYstem*, is specifically designed to facilitate scalability testing programmable data-plane technologies and network-function virtualization frameworks over standard telco scenarios.[1] TIPSY features

- a flexible and concise configuration subsystem that allows to set the *pipeline-size* scaling dimension easily,
- a configurable trace generator to produce deterministic traffic traces at the required *active flow count*,
- a set of reference telco pipeline implementations for different backends with configurable CPU core count for scaling along the *concurrency* dimension,
- and a simple evaluation framework to visualize results.

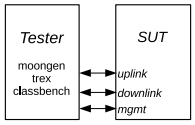


Fig. 4. TIPSY: architecture.

The target audience for TIPSY is network vendors and operators who want to evaluate the scalability of a programmable switch in increasingly complex configurations, DevOps teams that want to integrate a VNF pipeline into a continuous integration framework, operators certificating a 5G product against standard benchmarks, and researchers validating new algorithms/data-structures. A unique feature of TIPSY is that it allows *reproducible and dynamic benchmarks*; TIPSY is designed from the bottom up to ensure that repeating an experiment on the same system would always generate the same result (e.g., a `fake-drop` mode for controlling packet drops, deterministic packet trace generation, etc.) and it can trigger a controller to periodically modify and reconfigure some aspect of the static pipeline, like initiating a configurable number of user handover events in a 5G gateway, to obtain data-plane benchmarks under controller churn.

The TIPSY reference architecture is given in Fig. 4. A typical setup contains a SUT that is to be evaluated and a Tester that drives the measurements. The two systems are connected back to back with a pair of high-speed links, the *uplink* and the *downlink*, used by the Tester to feed the SUT with test traffic and measure the output, plus a management link that is used by the Tester to configure the SUT.

The TIPSY workflow is as follows (see Fig. 5). First, the user chooses a reference 5G pipeline and issues `tipsy init <pipeline>` to create a main configuration file (Fig. 6). This sample configuration may then be modified to set the scaling mode (in the `benchmark` section), the pipeline size as exposed by the specific pipeline (`pipeline` section), the active flow count and packet size (`traffic` section), and the layout of the required result charts (`visualize` section).

Scaling parameters may be specified as lists in brackets; in "joint" scaling mode subsequent values for each list will be selected at once (the first elements from each list, then the second elements, etc.) while in "outer" mode separate benchmarks are made for the Cartesian product of the lists. Benchmarks can be decomposed into multiple configuration files, which allows to factor site-specific settings into a platform-default configuration for easy reuse. Issuing `tipsy make` (or simply `make`) will perform the requested benchmarks; under the

[1]Code and documentation are available at https://github.com/hsnlab/tipsy.

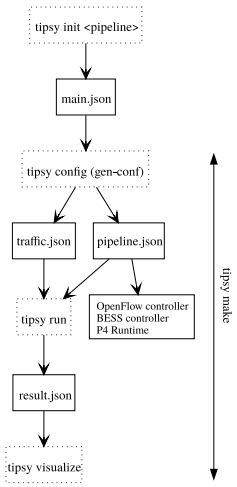


Fig. 5. TIPSY workflow.

```
{
  "benchmark": [{
     "scale": "outer",
     "pipeline": {
       "name": "mgw",
       "user": [1, 1000]
     },
     "traffic": {
       "pkt-num": 10,
       "pkt-size": [64, 128, 256]
     }
  }],
  "visualize": [{
    "type": "simple",
    "title": "RX {pipeline.name}",
    "group-by": "pipeline.user",
    "x-axis": "traffic.pkt-size",
    "y-axis": "out.throughput.RX.
         PacketRate"
  }]
}
```

Fig. 6. Sample TIPSY configuration.

hood TIPSY will create a separate configuration for each input data point, generate a traffic mix for each measurement, configure the SUT with the static pipeline, fire up a controller to apply dynamic load, execute the measurements and collect the results, and then create all the requested visualizations and generate a report. TIPSY implements a subset of the MongoDB query language and backend, which permits filtering results and generating charts for select aspects of the benchmark.

IV. STANDARD 5G TELCO PIPELINES

One of the stated goals of TIPSY is to fix representative telco pipelines that may serve as a reference for future benchmarking efforts. Currently, the benchmark suite contains 10 5G pipelines defined and 6 fully and 2 partially implemented (see Table I). The `PORTfwd` scenario represents a simple L2 repeater used to obtain the baseline performance, the `L2fwd` pipeline models an Ethernet switch, `L3fwd` implements an IP router with group-table processing, and Encap/Decap, RateLimit, Firewall, and NAT specify basic standalone VNF micro-benchmarks. The remaining 3 pipelines are complex 5G reference service chains built from the previous blocks.

A. The Mobile Gateway Pipeline

Due to space constraints we detail only the 5G mobile gateway reference pipeline (`mgw`); for a discussion of the data-center gateway and the broadband network gateway consult the TIPSY repository or [10], [34]. This pipeline connects 5G user equipment (or users), located behind base stations, to the public Internet (see Fig. 7).

Telco pipelines usually have a separate uplink (user-to-Internet) and downlink direction (Internet-to-user); TIPSY enforces this distinction in all pipelines. In the uplink direction the gateway receives GTP-encapsulated packets from the base stations, identified by the source IP address in the GTP header, and forwards the decapsulated packets to a set of destination hosts representing the Internet services accessed by the users. After decapsulation, the source IP address identifies the user and the GTP TEID identifies the bearer, and the destination IP address designates the public service. After various checks, the uplink pipeline decapsulates the packet from the GTP tunnel, identifies user equipment, applies policing, and finally routes the decapsulated packet to the Internet based on an L3 forwarding table. The downlink pipeline is basically the reverse; the gateway receives normal packets from the Internet, identifies the user equipment/bearer based on the packet destination IP address, rate-limits users' flow, and encapsulates and sends the packet to the appropriate base station.

The *static pipeline size*, both in the uplink and the downlink direction, can be scaled through multiple configuration knobs:

- `user`: the number of users generating uplink traffic;
- `bst`: the number of base stations;
- `server`: the number of public IP destinations;
- `nhop`: L3 group table size.

In addition, the following *dynamic* events can be applied to the static pipeline, with the parameter specifying the rate by which the respective event is initiated marked in parenthesis.

- *User update*: user arrival/departure event, which involves updating the user selector table and adding a new queue to the rate limiter (`fluct-user`).
- *Handover*: a user's attachment point changes, involving an update to the user selector table (`handover`).
- *Server update*: addition/removal of a server, updating the L3 FIB and the group table (`fluct-server`).

B. Backends

TIPSY adopts a modular architecture, which allows to freely add new pipelines, new packet generators, and new software switch backends. This design lets TIPSY to easily integrate *general purpose tools* to drive the measurements, like the `moongen` packet generator which implements a flexible scripting interface on top of LuaJIT, achieves line-rate packet rate for 10G and 40G NICs, and features nanosecond-precision latency measurements in a loopback measurement configuration using hardware timestamping [35], as well as *special purpose tools* like the `classbench-ng` framework to generate realistic ACLs and matching traffic traces for the Firewall micro-benchmark [36] and the `trex` traffic generator which features dynamic translation learning for the NAT benchmark (see Table I). The switch backends with most complete TIPSY support at the moment are as follows.

TABLE I

TIPSY REFERENCE PIPELINES, WITH AVAILABLE BACKENDS. COLUMN ENCAP/DECAP SPECIFIES THE TYPE OF TUNNELING SUPPORTED BY THE PIPELINE, PARSEFIELD GIVES THE TYPES OF HEADER FIELDS PARSED AND SETFIELD GIVES THE FIELDS MODIFIED BY THE PIPELINE, RATELIMIT, FIREWALL, AND NAT SPECIFY WHETHER THE PIPELINE CONTAINS THE RESPECTIVE VNF, BACKEND GIVES THE AVAILABLE IMPLEMENTATIONS

Pipeline	Features						Backends
	Encap/Decap	ParseField	SetField	RateLimit	Firewall	NAT	
Port forward (`PORTfwd`)	–	–	–	–	–	–	BESS, ESwitch, Lagopus, NetBricks*, OF–DPA, OVS, P4, Vpp
L2 forward (`L2fwd`)	–	L2	–	–	–	–	BESS, ESwitch, Lagopus, NetBricks*, OF–DPA, OVS, P4, Vpp
L3 forward (`L3fwd`)	–	L2/L3	L2/L3	–	–	–	BESS, ESwitch, Lagopus, NetBricks*, OF–DPA, OVS, P4, Vpp
Encap/Decap (`encap`)	VXLAN	–	L2/L3	–	–	–	–
RateLimit (`rlimit`)	–	–	–	x	–	–	–
Firewall (`acl`)	–	L2/L3/L4	–	–	x	–	BESS, ESwitch*, Lagopus*, OVS
NAT (`nat`)	–	L2/L3/L4	L2/L3/L4	–	–	x	ESwitch*
Data Center GW (`dcgw`)	VXLAN	L2/L3	L2/L3	–	–	x	OVS*
Mobile GW (`mgw`)	GTP	L2/L3/L4	L2/L3	x	–	–	BESS, ESwitch, Lagopus, OVS, P4
Broadband Network GW (`bng`)	GRE	L2/L3/L4	L2/L3	x	x	x	BESS, ESwitch, Lagopus, OVS, P4

*Implementation not yet released with TIPSY.

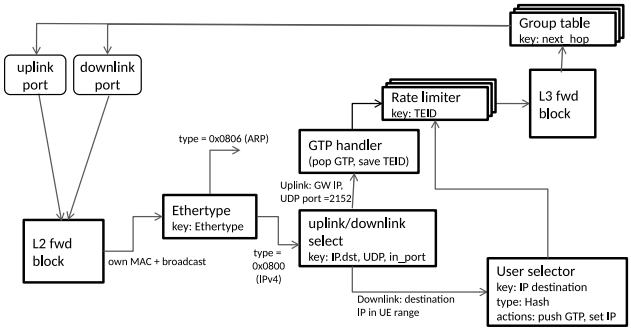


Fig. 7. The mobile gateway (`mgw`) pipeline.

- Open vSwitch (OVS, [9]): OVS is undoubtedly the most popular programmable software switch in use today. OVS can be configured via OpenFlow [7] or the OVSDB protocol [6] and it supports basically all popular L2/L3/L4 protocols and tunnel schemes. The OVS fast datapath is built on a sophisticated flow-cache hierarchy [9].
- Lagopus [37]: a high-performance software OpenFlow switch built with DPDK [33].
- The Berkeley Extensible Software Switch (BESS, [14]): an integrated SDN/NFV data plane that can be programmed through connecting simple functional modules into a VNF graph via a Python interface and/ or gRPC.
- ESwitch [10]: The Ericsson reference OpenFlow software switch specialized for telco use cases, featuring on-the-fly pipeline compilation for extremely fast performance.

The rest of the backends, OF–DPA (the OpenFlow–Data Plane Abstraction layer for the Broadcom Silicon SDK [17]), P4 (we used the `t4p4s` retargetable P4 compiler from the P4@ELTE project [12]), Vpp (the `fd.io` universal dataplane [16]), and NetBricks [15] are only partially supported, either because they do not provide all features necessary to implement every 5G pipeline or because they do not allow easy dynamic configuration from the TIPSY controller. Note that even the fully supported pipelines (Lagopus, OVS, BESS, and ESwitch) require backend-specific workarounds; e.g., only ESwitch supports GTP, BESS does not contain a rate-limiter so policing is applied through the scheduler and `t4p4s` does not provide data-plane rate-limiting functionality at all, etc. Additionally, some switch features were unused for reproducibility; e.g., even though both OVS and BESS contain a NAT we still used a "statically populated" NAT in order to maintain control over the NAT mappings, which we need in order to generate valid and reproducible downstream traffic.

TABLE II

SUT HARDWARE & SOFTWARE SPECIFICATIONS

SUT	12 × isolated Intel Xeon E5-2680 CPU@2.50GHz, 64Gbyte DRAM, 2 × Intel XL710 40GbE NIC, Linux 4.4.0, Ubuntu OS
BESS	v0.3.0-493-g2294e91 & DPDK 17.11
ESwitch	hg@971ad & DPDK 17.11
Lagopus	0.2.10-release & DPDK 16.11.0
OVS	v2.8.1 & DPDK 17.05
P4	`t4p4s` P4-16 git@eb6c9f & DPDK 17.11
Vpp	v18.01.1 & DPDK 17.11
OFDPA	Edge-Core AS4610-54T/ONL, Quanta T1048-LB9/PicOS

V. SCALABILITY STUDIES

Below we highlight some important findings from the scalability study we have conducted in order to evaluate the applicability of different programmable software switch suites for the next-generation 5G mobile core. The SUT configurations are given in Table II. All benchmarks were conducted with minimum-size (64-byte) packets using the RFC2544 evaluation methodology (maximum throughput and latency attainable with at 0.2 mpps drop rate and hardware packet timestamps); the configuration supports a maximum of 22.4 million packets per second (mpps) rate in this setup. TIPSY configurations are available in the github repository.

A. Switch Architecture Matters

First, we evaluated the single-core baseline performance of 6 programmable software switches, BESS [14], ESwitch [10], Lagopus [37], OVS [9], `t4p4s` [12], and Vpp [16]; see Fig. 8 for the raw packet rate and latency results. Since our testbed did not have SmartNICs available, we added two OF–DPA compatible hardware switches (a Quanta and an Edge-core device, see Table II) that represent the hardware offload option in the benchmarks. Note that these switches have 10 Gbps ports only whereas the software switches are evaluated at 40 Gbps; still, the results are on the same scale. The baselines were obtained in the `L3fwd` pipeline, as the number

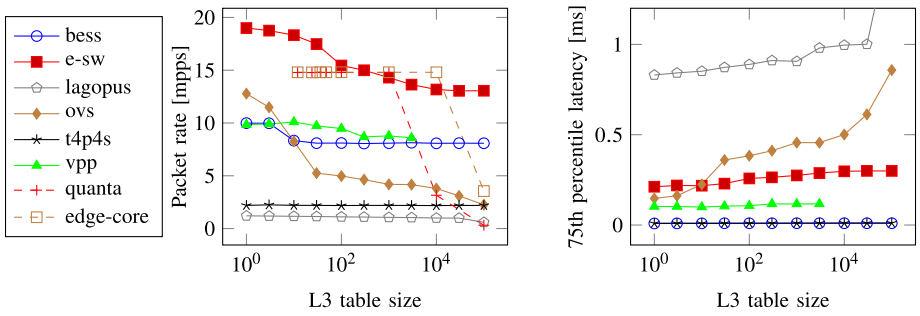


Fig. 8. Single-core baseline: raw packet rate in million packets per second (mpps) and 75th percentile latency in the `L3fwd` micro-benchmark at 64-byte packet size with BESS (`bess`), ESwitch (`e-sw`), Lagopus, OVS (`ovs`), P4 (`t4p4s`), Vpp (`vpp`), and the Quanta and the Edge-Core OF–DPA switches.

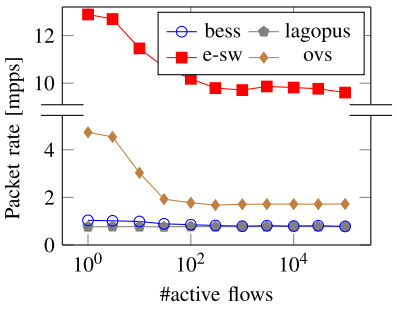


Fig. 9. Single-core throughput as the function of the active flow count (`mgw` pipeline: `bst=40`, `user=300`, `server=400`).

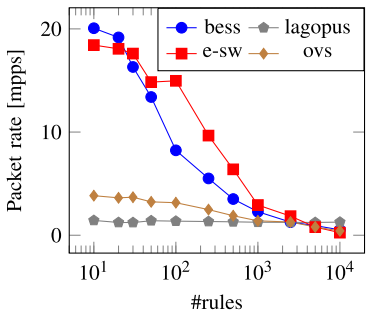


Fig. 10. Single-core performance in the Firewall (`acl`) micro-benchmark.

of entries in the L3 table (the static pipeline-size dimension) and the number of distinct destination IPs in the input trace (the active-flows dimension) were scaled jointly. Latency was measured at the RFC2544 packet rate, which likely contributes to the huge latency seen with ESwitch and OVS that produce by far the highest throughput.

Even in this micro-benchmark it stands out that *datapath architecture greatly affects scalability*. Hardware switches are essentially insensitive to workload size as long as the pipeline fits into the TCAM but show poor performance afterwards when forwarding falls back to the slow path; we expect similar behavior from SmartNICs. The software switches that wrap a high-performance longest-prefix-matching algorithm (ESwitch, BESS, `t4p4s`, Vpp) show similar scaling traits, producing consistently high throughput independently from table size. (The small decrease at higher workloads is due to CPU cache contention.) OVS, however, seems considerably sensitive to the workload size, with almost ten-fold throughput drop due to deteriorating flow-cache hit rate when the active flow count grows beyond a couple of thousands [9], [10].

We confirmed this finding in a complex macro-benchmark as well. Fig. 9 shows the throughput of OVS, BESS, and ESwtich on the mobile gateway pipeline (`mgw`) varying only the active flow count dimension while leaving the rest of the scalability parameters unchanged. The throughput of OVS drops by some 65% due to increasing traffic diversity and dropping flow-cache hit rate. This is particularly worrying: since the active flow count is the only scalability dimension that is usually not controlled by the mobile operator, even a single misbehaving user, or a legitimate IP scan, may lead to performance instability with OVS [9], [10]. Meanwhile, ESwitch shows only 25% degradation in the same setup.

Fig. 10 shows the performance in the Firewall (`acl`) micro-benchmark with increasingly complex realistic ACL rules and traffic mixes generated with `classbench-ng` [36]. Again, switch architecture largely determines performance: backends that implement ACLs with a specialized wildcard packet classifier (BESS and ESwitch) are faster than general-purpose rule-matching engines (OVS and Lagopus), but none of the switches scales beyond a couple of hundred firewall rules.

B. Pipeline Size Scalability is of Concern

Fig. 11 gives the single-core pipeline-size scalability for the mobile gateway (`mgw`) and the broadband network gateway (`bng`), in terms of downlink throughput measured for Lagopus, OVS, BESS, and ESwitch (the uplink charts were similar). As an indicator of the complexity of the pipelines the figures also specify the number of flow entries in the resultant OpenFlow tables.

Our results indicate that, as long as the pipeline size remains modest (below 100 OpenFlow table entries or the equivalent scale with BESS), performance figures are appealing with all throughput results ranging between 5–8 mpps (3–4 Gbps with minimum sized packets or 30–40 Gbps with maximum sized ones). However, as the number of subscribers scales into the range of a few thousands and pipeline complexity skyrockets (observe the log scale on the pipeline size axes), *all examined programmable switches exhibit poor pipeline scalability* with performance dropping 5 to 10-fold. ESwitch stands out though; thanks to the capability to dynamically compile a specialized datapath for each pipeline, it provides a high baseline single-core throughput and even though performance drops by 30% as the pipeline grows beyond

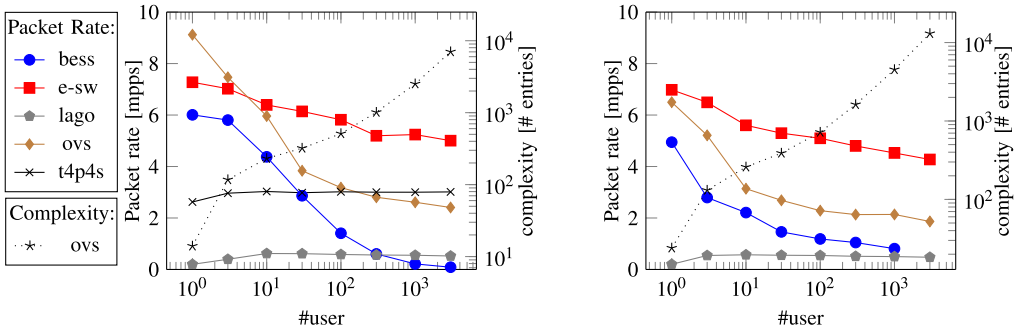


Fig. 11. Pipeline-size scalability: (a) the `mgw` and (b) the `bng` pipeline, downlink direction. Static pipeline complexity grows as follows: for `mgw` we set the unit scale at `bst=10`, `server=100`, and `user=3` and then we proportionally scale all three parameters *jointly*, and similarly for the `bng` case. Pipeline complexity is represented as the number of flow entries in the OVS OpenFlow pipeline.

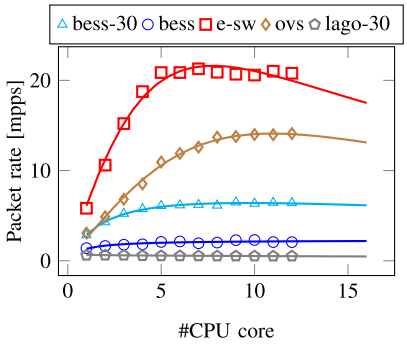


Fig. 12. Multi-core throughput and fitted USL (`mgw`), scaling all input dimensions *jointly* from `bst=30`, `server=300`, and `user=100`, plus a BESS result (`bess-30`) scaled from 30 users.

10,000 entries it still sustains roughly 4–6 mpps throughput with negligible packet loss.

C. Multi-Core Scalability is Challenging

After measuring scalability in the active flow-count dimension (Fig. 9) and the pipeline-complexity dimension (Fig. 11), we extend the analysis to the third scalability dimension: concurrency (see Fig. 12). These measurements were conducted by taking the settings `bst=30`, `server=300`, and `user=100` as the single-core unit scale and then increasing all three input dimensions *jointly* at the same pace. Note that omitting to scale the pipeline size and the active flow count dimensions one gets a false close-to linear multi-core scalability [29]. We added another measurement round for Lagopus and BESS this time starting from the 30-user baseline (`lago-30` and `bess-30`), since the 100-user result gave meaningless USL fitting.

Using the automatic USL-fitting feature of TIPSY, we get the following approximate USL parameters (see Fig. 12):

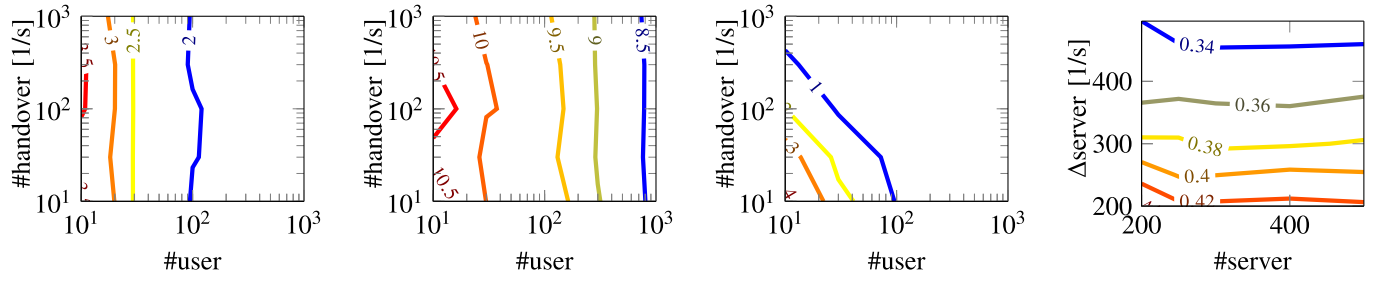
Switch	Base λ [mpps]	Contention σ	Crosstalk κ	n_{max}
OVS	2.5	0.002	0.008	10
BESS(30)	2.9	0.32	0.007	9
Lagopus(30)	0.6	1.09	0.020	2
ESwitch	6.1	0.06	0.016	5

Our results reveal fundamentally different multi-core scalability trends for each switch. Thanks to the sophisticated threading architecture [9], OVS attains very good multi-core scaling with close-to-zero contention ($\sigma = 0.002$) and very low crosstalk ($\kappa = 0.008$), reaching a maximum efficiency at 10 CPU cores. BESS, on the other hand, exhibits significant contention ($\sigma = 0.32$) between execution threads so that OVS, even executing consistently one order of magnitude larger pipelines, provides higher throughput when concurrency goes beyond 2 cores. Lagopus, furthermore, starts from the lowest baseline and reaches a performance cap already with 2 cores. For the first sight ESwitch parallel scaling looks the poorest; however, this conclusion is wrong since the SUT supports only 22.4 mpps maximum packet rate at 64-byte packets, and so ESwitch in fact already attains the physical limit with 5 cores. This result suggests that fitted USL models must be handled with extreme caution. In summary, most examined switches *fail to scale to more than 10 threads*; in fact, BESS performance starts to drop afterwards.

D. Dynamic and Static Load Evaluated Side-by-Side

So far, we have concentrated on static benchmarks, where performance is measured in steady state once the pipeline has been fully configured into the switch. In an 5G mobile core setup, however, the pipeline constantly changes; for instance as users enter and leave a cell, producing handover events, the pipeline needs to be updated with the user's new base-station association. Such reconfiguration events, especially if being generated multiple hundreds or thousands of times per second, may affect the packet processing performance adversely. Below, we measure how fast programmable switches can be reprogrammed and how performance scales with intensifying controller churn.

Fig. 13 gives the results for the `mgw` pipeline. In particular, the figures show the isolines for the measured throughput with OVS, BESS, and ESwitch, as the static complexity (the pipeline size in terms of the number of users) and the dynamic complexity (the intensity of `handover` events) change. The results for OVS (Fig. 13a) and ESwitch (Fig. 13b) indicate that *these programmable software switches exhibit extremely good dynamic scalability*, with both switches producing robust packet rate regardless of the frequency of handover events (vertical isolines). This indicates that OVS and ESwitch tolerate thousands of pipeline updates per second without major performance impact, although ESwitch provides consistently 4-times better throughput than OVS at equivalent scales. BESS results, on the other hand, seem less stable; for handover



(a) OVS (b) ESwitch (c) BESS (d) BESS

Fig. 13. Static vs. dynamic scaling: the uplink throughput (in mpps) `mgw` pipeline (`user=300`, `server=400`) as the function of the static pipeline size (number of users) and the dynamic workload (number of `handover` events per second) for (a) OVS, (b) ESwitch, and (c) BESS; and (d) BESS throughput as the function of the static L3 table size metric (`server`) and the intensity of the dynamic `server-update` events.

events (Fig. 13c) it shows roughly 1:1 rate between static and dynamic complexity (skewed isolines), while in the case of L3 update events (Fig. 13d) performance seems completely determined by the dynamic complexity (horizontal isolines) regardless of the static workload size.

We believe that again switch architecture is at play here: while ESwitch by design provides incremental and atomic pipeline updates with minimal data-plane impact [10], [38], BESS currently halts packet processing completely for the time match-action table modifications are accomplished.

VI. The Price for Data-Plane Programmability

As mobile vendors increasingly migrate 5G pipelines from fixed-function hardware offerings to virtual network functions, it has become essential to estimate the 5G workload scale a programmable software switch can handle economically and assess *how many CPU cores/blades are needed to achieve performance parity with a current fixed-function middlebox*. Below, we synthesize our results to answer these questions.

We chose the Broadband Network Gateway (BNG, [10], [34]) from the reference pipelines since it is relatively unchanged from 4G to 5G and thereby allows a comparison with a commercial fixed-function middlebox. The Ericsson SSR 8000 family of Smart Services Routers (SSR) supports $768,000$ users in the BNG configuration, each user receiving at least 3 Mbps access even with minimum-sized packets. The SSR-based BNG in full configuration is deployed on 12 2U-form-factor dual-socket blades in a single rack, consuming 14 kW of electricity, or 18 mW/user.

What is the scale of the deployment that could support the same load using OVS configured with the `bng` reference pipeline? First, suppose optimistically that OVS scales linearly in the concurrency dimension (which we know is not the case, cf. Fig. 12) and therefore we can extrapolate from the single-core measurements (Fig. 11b) to the full deployment. When configured with 300 users in the downlink direction of the `bng` pipeline (uplink results are similar), OVS runs at roughly 2 mpps packet rate on a single core, or about 1Gbps at 64-byte packets. This yields 3 Mbps/user, the same as the fixed-function BNG. At this scale, we would need 2,560 CPU cores to support 768k users with OVS, or roughly 100 blades (assuming 24 core/blade) in 7 racks (assuming 16 blades/rack). This is 8 times the deployment size of an equivalent SSR-based BNG, with an energy consumption of $\sim$ 50 kW (70 mW/user) assuming a conservative estimate of 500 W/blade. And this is an optimistic assumption, since OVS multi-core scaling is not linear in general (recall again Fig. 11b); using the fitted USL predictions *we get performance parity at roughly* 10*-times the scale of the SSR-based BNG*. With BESS as the software switch the scale would be almost twice as large. ESwitch on the other hand needs only 2 racks to run the same load as the SSR, thanks to that it can handle more users per core (Fig. 11b) with better multi-core scalability (Fig. 12).

TABLE III
Deployment Scale Needed to Handle the Load Equivalent to an Ericsson SSR (12 2U Blades in a Single Rack) Broadband Network Gateway (BNG, 768K Users, 3Mbps/User) and Mobile Gateway (MGW, 1 Million Users) With Programmable Switches

	Switch	Unit users/blade	Deployment Blades/Racks	Energy kW (mW/user)
BNG	BESS	4,320	178/12	89(116)
	OVS	7,200	107/7	54(70)
	ESwitch	24,000	32/2	16(21)
MGW	BESS	3,600	278/18	140(140)
	OVS	24,000	42/3	20(20)
	ESwitch	64,000	16/1	8(8)

Table III gives the results for the BNG and the MGW pipelines; note that for all results we used the optimistic linear-scaling assumption. For the MGW we took the Ericsson SSR Evolved Packet Gateway (EPG) as the baseline, supporting beyond 1 million users in the same form factor as the BNG. The EPG packet-processing pipeline, although functionally similar, is much more complex than our `mgw` pipeline (e.g., the EPG performs full-rate deep-packet inspection); still we believe that it serves as a useful comparison for the MGW use case. Again, we see that *with OVS and BESS we need* 3–12 *times as many blades as the EPG to support the same load, with only ESwitch reaching performance parity at roughly the same scale and with much smaller energy consumption.*

VII. Conclusions and Future Research

In this paper we report on a multi-year effort to analyze the applicability of current programmable software data-plane technology to implement the next-generation mobile core. We presented a taxonomy for data-plane scalability analysis

and we introduced TIPSY, a purpose-built benchmarking framework that provides simple declarative benchmark descriptions, automatic static and dynamic pipeline configuration, unassisted measurements, and configurable visualizations.

Out results indicate that *current programmable software switches struggle to scale to the same load as commercial fixed-function appliances*, which suggests that *one pays dearly for the added configurability, elasticity, flexibility, manageability, and deployment speed, of the programmable data plane*. The only exception seems to be ESwitch, which can handle roughly the same load as a fixed-function middlebox but with all the added benefits of programmability. For the moment, however, ESwitch remains a proprietary proof-of-concept prototype, with little hope to become available soon as an open source software or as a boxed commercial product.

We conclude that *scalability in the programmable software data plane is of concern*. This calls for further research; we believe that progress on the below topics would be particularly important to change the current state-of-affairs [39].

- *Software packet classification.* Our benchmarks revealed that packet classification remains by-far the most taxing operation in the programmable data plane [9], [10], [21]. This is because programmable switches generally support more complex classifier rules than conventional fixed-function appliances, which raises substantial theoretical and practical challenges.
- *Pipeline embedding.* Most nontrivial pipelines have several functionally equivalent forms and finding the best representation for a given software switch is a painful process at the moment. BESS, for instance, allows to decompose a complex pipeline into smaller chunks that can be effectively scheduled to a single CPU core, yet our current, admittedly naive, approach is to implement the whole pipeline as a single functional block and simply repeat this large block to all CPU cores. A sound theoretical framework to optimally compile a pipeline to the underlying software data-plane would be very useful here; see a similar approach for HW switches in [40].
- *Black-box switch tuning.* One of the most tedious steps in data-plane benchmarking is fixing a base SUT configuration for maximal efficiency; even a single Linux-kernel `sysctl` or a DPDK configuration knob can have dramatic effects on the performance of the pipeline that runs on top [26]. Using a black-box optimization tool, like Google Vizier, for unassisted switch fine-tuning promises itself a compelling research topic.

ACKNOWLEDGMENT

The authors would like to thank Justine Sherry and Barath Raghavan for the discussions that motivated the paper.

REFERENCES

[1] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 80–87, May 2017.

[2] K. Obraczka, C. Rothenberg, and A. Rostami, "SDN, NFV and their role in 5G," in *Proc. ACM SIGCOMM Tuts.*, 2016. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2016/tutorial-sdnnfv5g.php

[3] Z. A. Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker, "A high performance packet core for next generation cellular networks," in *Proc. ACM SIGCOMM*, 2017, pp. 348–361.

[4] C. E. Rothenberg *et al.*, "When open source meets network control planes," *Computer*, vol. 47, no. 11, pp. 46–54, Nov. 2014.

[5] A. Manzalini, "Towards 5G software-defined ecosystems," IEEE Softw. Defined Netw., White Paper, 2016. [Online]. Available: https://sdn.ieee.org/publications/towards-5g-software-defined-ecosystems

[6] B. Pfaff and B. Davie, *The Open vSwitch Database Management Protocol*, document RFC 7044, 2013.

[7] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[8] C. Zhang, S. Vasudevan, and S. Wong, "Extensible Neutron service function chaining—Here it comes," in *Proc. OpenStack Summit*, 2015. [Online]. Available: https://www.youtube.com/watch?v=sndi0QGAYUc

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

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

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

[12] S. Laki, D. Horpácsi, P. Vörös, R. Kitlei, D. Leskó, and M. Tejfel, "High speed packet forwarding compiled from protocol independent data plane specifications," in *Proc. ACM SIGCOMM Demo*, 2016, pp. 629–630.

[13] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," in *Proc. USENIX NSDI*, 2014, pp. 445–458.

[14] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "SoftNIC: A software NIC to augment hardware," Dept. EECS, Univ. California Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2015-155, 2015.

[15] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," in *Proc. USENIX OSDI*, 2016, pp. 203–216.

[16] *The FD.io Project*. Accessed: Mar. 12, 2018. [Online]. Available: https://fd.io

[17] *The OF-DPA Project*. Accessed: Mar. 11, 2018. [Online]. Available: https://github.com/Broadcom-Switch/of-dpa

[18] M. Holdorf, "How-to compare performance of data plane devices," in *Proc. Netw. Archit. Services*, 2016, pp. 33–40.

[19] R. V. Rosa and C. E. Rothenberg, "Taking open vSwitch to the Gym: An automated benchmarking approach," in *Proc. IETF/IRTF Workshop*, 2017, pp. 1–14.

[20] L. Csikor, M. Szalay, B. Sonkoly, and L. Toka, "NFPA: Network function performance analyzer," in *Proc. IEEE NFV-SDN*, Nov. 2015, pp. 17–19.

[21] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: Data plane performance," in *Proc. IEEE ICC*, May 2010, pp. 1–5.

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

[23] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation," in *Proc. PAM*, 2012, pp. 85–95.

[24] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet IO," in *Proc. ANCS*, 2015, pp. 29–38.

[25] J. Blendin, Y. Babenko, D. Kusidlo, G. Schyguda, and D. Hausheer, "Towards a structured approach to developing benchmarks for virtual network functions," in *Proc. EWSDN*, 2016, pp. 1–6.

[26] P. Zhang, "Configuring and benchmarking open vSwitch, DPDK and vhost-user," in *Proc. KVM Forum*, 2017.

[27] *IxNetwork Overview: L2/3 Network Infrastructure Performance Testing*, IXIA, Calabasas, CA, USA, 2017.

[28] R. Durner, A. Blenk, and W. Kellerer, "Performance study of dynamic QoS management for OpenFlow-enabled SDN switches," in *Proc. IEEE 23rd Int. Symp. Qual. Service (IWQoS)*, Jun. 2015, pp. 177–182.

[29] B. Schwartz, *Practical Scalability Analysis With the Universal Scalability Law*. Charlottesville, VA, USA: VividCortex, 2015.

[30] S. Bradner and J. McQuaid, *Benchmarking Methodology for Network Interconnect Devices*, document RFC 2544, 1999.
[31] N. J. Gunther, *Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services*, 1st ed. Berlin, Germany: Springer-Verlag, 2010. [Online]. Available: https://www.springer.com/gp/book/9783540261384
[32] P. Shinde, A. Kaufmann, T. Roscoe, and S. Kaestle, "We need to talk about NICs," in *Proc. USENIX HotOS*, 2013, p. 1.
[33] Intel. *Data Plane Development Kit*. Accessed: Mar. 12, 2018. [Online]. Available: http://dpdk.org
[34] Intel. *Network Function Virtualization: Quality of Service in Broadband Remote Access Servers With Linux and Intel Architecture*. Accessed: Feb. 16, 2014. [Online]. Available: https://networkbuilders.intel.com/docs/Network_Builders_RA_NFV_QoS_Aug2014.pdf
[35] P. Emmerich, S. GallenmÃ¼ller, D. Raumer, F. Wohlfart, and G. Carle. (2014). "MoonGen: A scriptable high-speed packet generator." [Online]. Available: https://arxiv.org/abs/1410.3322
[36] J. Matoušek, G. Antichi, A. Lučanský, A. Moore, and J. Kořenek, "ClassBench-ng: Recasting classbench after a decade of network evolution," in *Proc. ANCS*, 2017, pp. 204–216.
[37] Y. Nakajima, T. Hibi, H. Takahashi, H. Masutani, K. Shimano, and M. Fukui, "Scalable high-performance elastic software OpenFlow switch in userspace for wide-area network," in *Proc. Open Netw. Summit (ONS)*, Santa Clara, CA, USA, 2014, pp. 1–2.
[38] J. Han *et al.*, "Blueswitch: Enabling provably consistent configuration of network switches," in *Proc. ACM/IEEE ANCS*, 2015, pp. 17–27.
[39] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *Proc. IEEE HPSR*, Jun. 2018.
[40] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *Proc. USENIX NSDI*, 2015, pp. 103–115.

Tamás Lévai received the M.Sc. degree in computer engineering from the Budapest University of Technology and Economics in 2016, where he is currently pursuing the Ph.D. degree. His research interest focuses on software-defined networking and high-performance packet processing.

Gergely Pongrácz graduated from the Technical University of Budapest in 2000. In 2004, he became a Research Engineer at Ericsson Research, where he is an expert in researching programmable data plane. He is involved in NFV and SDN topics, especially in the programmable networking area. These projects resulted in well received papers and demos, such as a paper on the IEEE SigComm in 2016 or demos at the Mobile World Congress in 2015 and 2017.

Péter Megyesi received the Ph.D. degree in computer science from the Budapest University of Technology and Economics and the Doctoral School on Innovation and Entrepreneurship organized by the EIT Digital. He co-founded LeanNet, a Hungary based start-up focused on integrating software-defined networking into leading cloud native solutions, where he currently serves as a Chief Technology Officer.

Péter Vörös received the M.Sc. degree from the Doctoral School of Computer Science, Eötvös Loránd University, Budapest, Hungary, in 2014, and graduated from the Doctoral School of Computer Science, Eötvös Loránd University, in 2017. He is currently pursuing the Ph.D. degree with the Department of Information Systems, Eötvös Loránd University. He is also an Assistant Lecturer with the Department of Information Systems, Eötvös Loránd University. He is currently working on the projects in network security, traffic analytics, and programmable data planes.

Sándor Laki received the M.Sc. and Ph.D. degrees in computer science from Eötvös Loránd University in 2007 and 2015, respectively. He is currently an Assistant Professor with the Department of Information Systems, Eötvös Loránd University. He has authored over 20 peer-reviewed papers and demo papers, including publications at JSAC, INFOCOM, ICC, and SIGCOMM. His research interests include active and passive network measurement, traffic analytics, programmable data planes, and their application for new networking solutions.

Felicián Németh received the M.Sc. degree in computer science from the Budapest University of Technology and Economics (BME) in 2000. He is currently a Research Fellow with the Department of Telecommunications and Media Informatics, BME. He was a member of national research projects and the EFIPSANS, OPENLAB, and UNIFY EU projects. His current research interests include different aspects of 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 Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics. 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. He maintains several open-source scientific tools written in Perl, C, and Haskell.