Comparing the Performance of State-of-the-Art Software Switches for NFV

# Comparing the Performance of State-of-the-Art Software Switches for NFV

Tianzhu Zhang
LTCI, Telecom Paris
Institut Polytechnique de Paris
France

Leonardo Linguaglossa
LTCI, Telecom Paris
Institut Polytechnique de Paris
France

Massimo Gallo
Nokia Bell Labs
Paris, France

Paolo Giaccone
Politecnico di Torino
Turin, Italy

Luigi Iannone
LTCI, Telecom Paris
Institut Polytechnique de Paris
France

James Roberts
LTCI, Telecom Paris
Institut Polytechnique de Paris
France

## ABSTRACT

Software switches are increasingly used in network function virtualization (NFV) to route traffic between virtualized network functions (VNFs) and physical network interface cards (NICs). Understanding of alternative switch designs remains deficient, however, in the absence of a comprehensive, comparative performance analysis. In this paper, we propose a methodology intended to be fair and use it to compare the performance of seven state-of-the-art software switches. We first explore their respective design spaces and then compare their performance under four representative test scenarios. Each scenario corresponds to a specific case of routing NFV traffic between NICs and/or VNFs. Our experimental results show that no single software switch prevails in all scenarios. It is therefore important to choose the one that is best adapted to a given use-case. The presented results and analysis bring a better understanding of design tradeoffs and identify potential bottlenecks that limit the performance of software switches.

## 1 INTRODUCTION

For many years developers have used software packet processing for fast prototyping and functional testing but have relied on the superior performance of proprietary hardware for product deployment. The limitations of commodity off-the-shelf (COTS) servers, whose general-purpose kernels and chips were not optimized for packet processing, outweighed the flexibility advantage of software solutions. This situation has changed in recent years, thanks largely to the impulsion of Software-Defined Networking (SDN) and Network Function Virtualization (NFV) but also due to advances in the performance of COTS hardware. It is now widely accepted that significant savings in both CapEx and OpEx can be realized on replacing expensive, proprietary and inflexible hardware middleboxes by software counterparts.

A major spur to progress has been the development of high-speed I/O frameworks (e.g., DPDK [5], PF_RING ZC [10], and netmap [54]) that employ acceleration techniques, like kernel bypass and batch processing, to achieve performance comparable to that of proprietary hardware appliances. Software switches have been developed to bridge and route traffic in SDN and NFV and their performance has largely benefited from the use of these acceleration techniques. These developments are typified by state-of-the-art proposals such as FastClick [21], Open vSwitch (OvS) [8], Vector Packet Processing (VPP) [16], and the Snabb NFV project [13].

While interest in software switches is soaring, the relative merits of different proposals are still not well-understood in the absence of a comprehensive, comparative performance analysis. It is indeed a daunting task to perform such an evaluation [32] and most published comparisons relate to a small number of switch proposals [30, 58] or execute a limited number of test scenarios [49]. The objective of the present work is to propose a methodology for comparing switch performance, in terms of essential throughput and latency metrics, that takes proper account of the disparate design choices that guided their development. For instance, OvS was tailored to support match/action semantics, VPP was constructed as a full-fledged software router, while other solutions such as Snabb, FastClick, and BESS embraced modular design to compose complex network services.

To give a sense of the complexity of the task, we consider a simple scenario commonly used for performance comparisons. In this scenario, a software switch is deployed as an L2 forwarder between two 10 Gbps network interface cards (NICs). Bidirectional maximum throughput and the corresponding round-trip latency (with an offered input traffic load equal to 95% of the maximum throughput) are measured with minimum size, 64B packets. Fig. 1 reports some of our experimental findings. In the left-hand figure, we plot measured throughput against average latency. The metrics are seen to be negatively correlated, meaning the switch with the highest throughput is also the one achieving the lowest latency. In the right-hand figure, we plot the standard deviation of latency against its mean. However, this time no pattern is visible.

Based on these observations, we conclude that it is critical to fairly compare a broad range of state-of-the-art software switches in a set of simple yet representative test scenarios. To define the methodology, we start by analyzing the design space of seven software switches, namely Open vSwitch DPDK (OvS-DPDK) [9], FastClick [21], Berkeley Extensible Software Switch (BESS) [38], netmap suite [40, 47, 54, 55], Snabb [49], t4p4s [42], and FD.io VPP [16], to build a basic understanding of their respective designs. We then define four test scenarios, introduced in [62] and intended

*CoNEXT '19, December 9–12, 2019, Orlando, FL, USA*

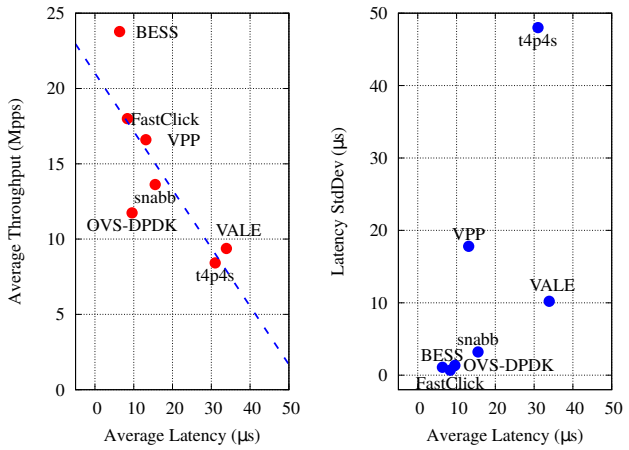



**Figure 1: Scatter plots of latency/throughput and of average/standard deviation of latency, under 64B synthetic packets and bidirectional 10Gbps links.**

to provide results meaningful for different segments of an NFV service chain. Finally, we provide experimental measurements of throughput, with unidirectional and bidirectional traffic, and latency. It is important to note that these experimental results depend significantly on the particular hardware and software versions used in our platform and are thus only indicative (for instance, VPP achieves higher performance under the FD.io Continuous System Integration and Testing (CSIT) tests [3] using a similar hardware configuration). Our aim is not, therefore, to discover the best performing solution for our hardware platform, but rather to define a proper comparison methodology and to identify possible performance impairments when the switches are used in the context of NFV. In our experiments, VNFs are hosted in virtual machines (VMs). The same tests can be repeated for other virtualization techniques such as containers, and we leave this for future work.

To facilitate reproducibility, all the scripts and instructions of our experiments have been released on GitHub [2]. We strongly encourage researchers and developers to use these to repeat the same set of experiments on their own servers and to build on this basis to gain further understanding.

The paper is organized as follows. In Sec. 2, we review related literature on software switches and their comparison. Then, in Sec. 3, we explore the design space of the considered software switches and highlight their specificities. In Sec. 4, we explain the four test scenarios. Experimental results are presented and discussed in Sec. 5. We draw our conclusions and discuss future work in Sec. 6.

## 2 RELATED WORK

We first survey the panoply of open-source software switch proposals before discussing related work on performance comparison of different implementations.

### 2.1 Software switches

We first introduce the seven switches whose performance we have directly compared and then briefly describe alternative designs.

*2.1.1 Evaluated Software Switches.* The seven state-of-the-art software switches included in our comparison study are OvS-DPDK, t4p4s, FastClick, Snabb, BESS, VPP, and VALE. They have been chosen both for the availability of an up-to-date codebase and for their promised high performance.

**OvS-DPDK** [9] is a high-speed user-space variant of Open vSwitch [8]. It moves the data plane of Open vSwitch into user space and adopts DPDK poll-mode drivers to deliver packets, completely avoiding the overhead imposed by kernel stack and interrupt handling.

**t4p4s** [42] is a platform-independent software switch specifically designed for P4 [25]. A compiler is implemented to generate switching code from P4 programs and a hardware abstraction layer deals with platform-dependent details. For Intel NICs, t4p4s integrates DPDK to improve efficiency.

**FastClick** [21] extends the codebase of Click Modular Router [41] and integrates high-speed packet I/O techniques such as DPDK and netmap. Its data path is also optimized by leveraging acceleration techniques including zero-copy, batching, and multi-queueing.

**Snabb** [49] is a high-speed modular software switch with a set of predefined modules enabling the composition of complex network functions. Like MoonRoute [34], it is based on Lua and LuaJIT [14]. Snabb is known for the introduction of the vhost-user protocol [15], featuring direct packet delivery between user-space processes and VMs, without kernel intervention.

**BESS** [38] is a modular software switch from UC Berkeley featuring a set of built-in modules used to compose network services. Modules can be glued together and fed to the daemon process, which deals with packet scheduling (enabling traffic prioritization) and processing.

**VPP** [16] is a software router that allows users to configure the forwarding graph and to process packets in batches. The VPP design incorporates a number of throughput optimization techniques while also supporting interrupt mode when using native drivers. In addition, VPP provides a CLI for experimentation and debugging while resorting to the binary API for production use (bindings to C, C++, Go, Python over a non-blocking, shared memory interface).

**VALE** [55] is an L2 software switch based on the popular netmap high-speed packet I/O framework. It adopts batch computing and memory prefetching to enhance processing efficiency. mSwitch [40] augments VALE with enhanced switching logic. To ensure high-speed packet delivery between virtual machines, a pass-through approach named *ptnet* is proposed [47]. In contrast to most of the other switches that use the DPDK poll-mode driver and complete kernel bypass, VALE is built on top of netmap and relies on system calls and NIC interrupts for packet I/O. It is therefore interesting to compare VALE with other solutions.

*2.1.2 Other Software Switches.* We briefly reference some other software switches that we have not included in the present comparison.

**RouteBricks** [28] achieves multi-Gigabit/s packet processing speeds by exploiting parallelization both within and across commodity servers. **PacketShader** [39] boosts packet processing using graphics processing units (GPUs). **Hyper-Switch** [53] improves packet forwarding between virtual machines and the Xen hypervisor by adopting batch processing and computation offloading. **Cuckoo Switch** [63], a software Ethernet switch, adopts the cuckoo hashing algorithm for forwarding table lookup and DPDK for packet I/O operations, thus realizing both memory efficiency and high-speed processing. **MoonRoute** [34] is a software router based on MoonGen [29] and LuaJIT [14]. The use of the Lua scripting language improves programmability compared to other software switches that use C or C++. Despite their interesting features, we exclude these switches from direct quantitative comparison because their codebase is quite outdated.

The Virtual Filtering Platform (**VFP**) [33] is designed to host SDN in cloud datacenters. **PVPP** extends VPP to support P4. Promising performance is reported in [27]. These solutions cannot be included for direct comparison as their code is not currently available.

**PISCES** [57] extends Open vSwitch with the support of the P4 language. However, as detailed in [42], t4p4s outperforms it by two times running the baseline L2 forwarding application. We thus only consider t4p4s in our comparison. **Lagopu**s [52] is a user-space OpenFlow switch based on DPDK. We do not take this switch into consideration here due to its low performance.[1] In [24], the **OfSoftSwitch** is enhanced by leveraging the PFQ framework [23]. However, like Lagopus, this switch has limited performance ($\leq$ 4 Mpps with 64B packets) and is therefore not included in our comparison.

Finally, **ClickNF** [35, 43] extends Click with a set of modules enabling complex L2 to L7 network functions. Since ClickNF is similar to FastClick in terms of design and performance, we do not consider it to avoid duplicates.

## 2.2 Performance Comparison

The literature includes a number of works aiming to evaluate the performance of state-of-the-art software switches. The authors of [31] compare Open vSwitch throughput with Linux bridge and Linux kernel IP forwarding. According to their results, the standard Open vSwitch fails to achieve 2 Mpps with 64B packets. The same authors further analyze the throughput and latency of Open vSwitch in [30]. Paper [58] presents an evaluation of OvS-DPDK throughput using port/flow mirroring with 1 Gbps NICs. Our work differs in that we only focus on software switch implementations capable of achieving much better performance (e.g. more than two orders of magnitude higher throughput).

Several prior performance comparison works in the literature are particularly relevant to ours. The survey [44] presents a throughput and CPU utilization performance comparison of SR-IOV, netmap passthrough, OvS-DPDK, and Snabb under two test scenarios: inter-VM forwarding and 1-VNF loopback. Our work differs in that we consider more software switches under a more diverse set of test scenarios (including those considered by the survey). Moreover, since our work focuses solely on software switches, we attach physical NICs to the VALE switch, not directly to the VMs. We also omit hardware PCI passthrough techniques such as SR-IOV.

The authors of [49] compare the throughput of Snabb, Open vSwitch, OvS-DPDK, and Linux bridge while [32] evaluates the throughput of BESS, VPP, and OvS-DPDK using physical interfaces only. Our work focuses as well on NFV use cases and thus also considers VNFs hosted in a virtualized environment. In addition, all the aforementioned works do not compare performance in the important scenario of service chains with more than one VNF. This multi-VNF loopback scenario is considered in [50, 51], but the comparison is limited to VPP and OvS-DPDK.

None of the above-cited works consider latency, which is a critical performance metric. Both throughput and latency of BESS and ClickOS are compared in [48] under a loopback service chaining scenario. We preferred to consider VALE, rather than ClickOS, in our comparison as the latter is a full-fledged NFV framework rather than a software switch. Furthermore, in our comparison, all systems use the same QEMU hypervisor, avoiding the uncertainty arising when one system uses QEMU and the other Xen.

In contrast to the existing literature, in addition to providing measurement results, our work seeks to define a comparison methodology. This consists of a set of test scenarios and metrics designed to enable a deeper understanding of software switch performance and to help identify potential bottlenecks. There are two open-source projects, namely FD.io CSIT-1904 [4] and VSperf [18], that are very relevant to our work. CSIT-1994 aims at defining a comprehensive set of test scenarios for VPP and DPDK applications. VSperf, proposed by the Open Platform for NFV Project (OPNFV), focuses on the benchmarking methodology of virtual switches for the NFV infrastructure [59]. Currently, it has integrated vanilla OvS, OvS-DPDK, and VPP. Our work covers all the test scenarios defined by the two projects. Moreover, the reported experimental results relate to a set of representative, state-of-the-art software switches that is more extensive than any considered in prior work.

# 3 SOFTWARE SWITCHES DESIGN SPACE

We first discuss the importance of exploring the different design objectives of alternative software switches before considering how the seven representative state-of-the-art solutions fit into a design space taxonomy.

## 3.1 Design objectives

Before performing a comparative evaluation, it is very important to understand the main design differences between the considered software switches. This may require identifying the adopted processing model, or ascertaining whether the switch has been designed for a particular application such as SDN or NFV. This is time-consuming but appears an essential precondition to avoiding biased results or an incorrect interpretation of the impact of subtle, performance impacting details.

Rather than providing a detailed discussion of implementation and/or acceleration techniques, for which we refer to the survey in [45], we aim in this section to consider each switch design in relation to a number of technical aspects affecting packet processing performance. The objective is to gain insight on how to devise meaningful experimental scenarios. A summary of this taxonomy

[1] Our preliminary benchmarking result with Lagopus and Ryu controller achieves a throughput less than 2 Mpps with 64B packets

**Table 1: Taxonomy of State-of-the-Art High-Performance Software Switches**

| | Architecture | | Programming | Processing Model | | Virtual | Runtime | Programming | Main |
|---|---|---|---|---|---|---|---|---|---|
| | Self-contained | Modular | Paradigm | RTC | Pipeline | Interface | Reprogrammability | Language | Purpose |
| BESS | | ✓ | Structured | ✓ | ✓ | vhost-user | Medium | C, Python | Programmable NIC |
| Snabb | | ✓ | Structured | | ✓ | vhost-user | High | Lua, C | VM-to-VM |
| OvS-DPDK | ✓ | | Match/action | ✓ | | vhost-user | Medium | C | SDN switch |
| FastClick | | ✓ | Structured | ✓ | | vhost-user | Low | C++ | Modular router |
| VPP | ✓ | | Structured | ✓ | | vhost-user | Medium | C | Full router |
| VALE | ✓ | | Structured | ✓ | | ptnet | Low | C | Virtual L2 Ethernet |
| t4p4s | ✓ | | Match/action | ✓ | | vhost-user | Medium | C, Python | P4 switch |

and the corresponding software switch classification is shown in Table 1.

## 3.2 Architecture

A major difference between software switches derives from the way packet processing is configured and, more importantly, executed. A *self-contained* architecture is defined as a full-fledged software that can be deployed with minimal configuration effort. The switch data path is predefined (though modifications at compile time are allowed) and all processing functions are deployed in a single process. In contrast, a *modular* architecture targets a high degree of flexibility. This is usually achieved by providing a set of predefined, well-known network functions that can be arranged in a forwarding graph. The latter can even be re-configured at run-time, when each node is a different thread or process, or extended with custom network functions.

Our evaluation takes into account four switches designed with a self-contained architecture: VALE [55], VPP [20], t4p4s [42], and OvS-DPDK [9]. VALE is an L2 learning switch based on netmap. It can interconnect both physical NICs and virtual interfaces and forward packets at high speed. Though it is feasible to connect VALE with an external program, it is considered here as a self-contained architecture. VPP consists of a forwarding graph with hundreds of functions with support for additional plugins [46]. It exposes a command-line interface that can be used to configure the router with a syntax similar to the Cisco IOS operating system. OvS-DPDK is a software switch built for SDN in which packet processing is realized via a set of match/action tables (cf. Sec. 3.3), which can be modified via the `ovs-vsctl` API. Custom packet processing can be realized by adding new code that must be compiled inside the original codebase. t4p4s is designed to support P4 [25] semantics, whose workflow is quite similar to OvS-DPDK. It consists of a parsing stage on packet entry and a de-parsing stage when packets exit. Match/action tables, described through P4, are deployed between these two stages to indicate the sequence of operations to perform on packets.

The other switch designs considered in our study, FastClick [21], BESS [38], and Snabb [49], belong to the modular category. FastClick, one of the latest versions of the original Click Modular Router, consists of a set of nodes that can be arranged using a Click-specific configuration language. BESS also has a modular architecture, although the modules are more general and less specialized than those of FastClick. Similarly, Snabb interconnects modules with links to compose network service.

## 3.3 Design paradigm

Software switch implementations are heavily influenced by their target use cases. We classify the design paradigm into two categories. The first one adopts structured programming to route traffic across VNFs, as done by a majority of existing software switches. The second solution is to use the match/action programming paradigm exploited by OvS-DPDK and t4p4s. Packet processing is realized using built-in packet classification algorithms that match specific header fields and apply the corresponding actions.

## 3.4 Processing model

When packets are delivered to a software switch, there are generally two ways to process them: *run-to-completion* (RTC) and *pipeline*. The former refers to a model in which a single thread performs full packet processing before being forwarded or discarded, while the latter refers to a model that packets go through several threads, each contains a portion of processing logic, to complete full processing.

Most frameworks (VPP, OvS-DPDK, t4p4s, and VALE) adopt the run-to-completion model to reduce the context switching overhead. Even FastClick, though being an extension of Click which was designed with a pipeline model in mind, has completely moved to a full run-to-completion approach. Snabb is the only considered switch that processes packets solely according to a pipeline model while BESS can adopt either model depending on the implemented multicore approach.

## 3.5 Virtual interfaces

Software switches rely on virtual interfaces to interact with VNFs and steer traffic on NFV platforms. Most of the VMs under QEMU /KVM communicate with the outside world using the virtio [56] standard. It consists of the `virtio_net` paravirtualized frontend network driver and the `vhost_net` backend driver. Traditionally, vhost_net takes packets into the kernel and copies them back to the user-space software switch. However, this is not desirable from a performance point of view. To address this issue, Snabb implements `vhost-user`, a backend driver allowing direct packet exchange between software switches and VMs. Compared with vhost_net, vhost-user provides better performance as it eliminates the overhead imposed by the kernel. DPDK also adopts this solution and hence all of the frameworks considered in this work, except VALE that is based on netmap, use `vhost-user` [15] as backend driver.

VALE adopts `ptnet` for efficient VM networking. `ptnet` is a new paravirtualized device driver that grants the VMs direct access to packet buffers of netmap ports on the host using the netmap API. Compared with vhost-user, ptnet delivers packets in zero-copy manner without incurring the overhead of queueing (as for virtio)

or packet descriptor format conversion, at the cost of a lower degree of host-VM isolation and more difficult live migration.

### 3.6 Runtime reprogrammability

Although software switches are usually easy to program, it is also important to consider their degree of *reprogrammability*. As an example, programmable packet processors can be written as a simple C program. However, adding a new feature may require rewriting part of the code and, sometimes, to also rerun, recompile or replace binary executables. However, a highly reprogrammable software switch may offer the possibility to change behavior directly at runtime, without the need for recompilation. We categorize the switches into three degrees of reprogrammability: high, medium and low.

Snabb and BESS are the software switch implementations with the highest degree of reprogrammability. Thanks to the App engine and command-line tools of Snabb, standard modules can be loaded interactively and the processing pipeline can be dynamically adjusted at runtime. Similarly, BESS provides a control utility (`bessctl`) capable of loading new configuration files and modules into its processing pipeline on the fly. Note that newly added modules do need to be compiled before being integrated into the processing pipeline. OvS-DPDK packet processing behavior can also be adjusted at runtime. In particular, external controllers can populate flow rules to the OvS match/action tables through southbound protocols such as OpenFlow. As a result, its runtime programmability really depends on that of the external control plane. Both VPP and FastClick allow to program some modules and execute custom packet processing applications. In particular, the VPP command-line interface allows existing modules to be configured and new plugins to be added at runtime. Yet changing the version of the same plugin requires restarting the software switch. Therefore, VPP has a medium degree of reprogrammability. Similarly, even though some modules can be interactively configured, FastClick instance has to be restarted when the processing graph is changed and therefore has a medium degree of reprogrammability. Finally, both t4p4s and VALE switch have a low degree of reprogrammability since they do not provide any means to dynamically adjust their packet processing at runtime.

### 3.7 Programming language

The choice of one particular programming language over another may be dictated by performance requirements, programmability, or time-to-market considerations. Most of the software frameworks for high-speed packet processing are written in C and/or C++. Since both languages are performant, feature-rich, and portable across different platforms, most of the software switches considered in our study implement their performance-critical components using them. Higher-level programming languages such as Python and Lua are also used by some software switches. For example, BESS additionally provides a Python API to facilitate the composition of configuration scripts, t4p4s implements its P4 compiler in Python. Snabb, on the other hand, is based on Lua. It also wraps snippets of C code using LuaJIT, which profiles and optimizes code execution at runtime [49]. With the relatively better programmability of Lua and dynamic optimization of LuaJIT, Snabb is expected to be an efficient solution.

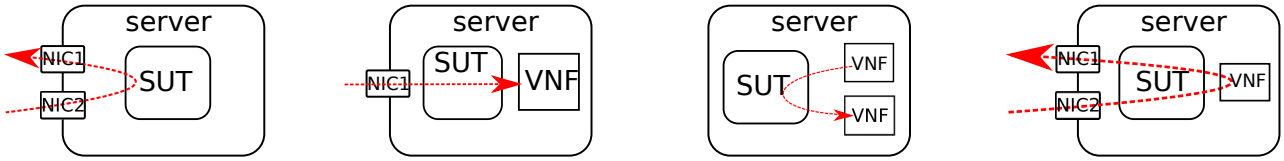
### 3.8 Switch main purpose

Packet processing frameworks are able to sustain good performance, thanks to a large collection of acceleration techniques discussed in the survey [45]. The adoption of these techniques depends on the main purpose for which the software switch has been designed. Considering this purpose is of interest for two main reasons: (i) it may provide hints on the performance of each design in some specific scenarios; (ii) it may be helpful in understanding which of the software switch implementations is more suitable for some particular user requirements.

BESS provides a native way to easily schedule packets without only using the simple FIFO approach, thus enabling custom policies, resource sharing, and traffic shaping. Resource sharing mechanisms may also be implemented on top of existing frameworks: e.g., Addanki et al. [19] implemented fair sharing of both CPU and bandwidth using fair packet dropping on top of VPP. However, to the best of our knowledge, BESS is the only design that natively provides scheduling capabilities without the need to write a custom algorithm. Snabb targets a simple and performant packet processing framework. Its core optimizations leverage runtime profiling and rely on LuaJIT to optimize the most frequently executed portion of code, rather than relying on the static compilation. Its app engine can dynamically register new apps, making it one of the most flexible solutions for high-speed packet processing. Unlike other switches, It implements its own compact kernel bypass mechanism without relying on DPDK or netmap. OvS-DPDK aims to provide the benefits of SDN (i.e., separation of data and control planes) with the flexibility of a software solution. Its data path is highly optimized thanks to the presence of internal flow caches. It can also be used as a static switch with predefined rules, or as a fully functional SDN switch in conjunction with an external control plane. t4p4s implements a high-speed, platform-independent P4 switch. Its compiler synthesizes P4 programs and generates core switch code, which is then converted to platform-specific instructions by its hardware abstraction layer. It is representative of several efforts to implement production-ready P4 switches. FastClick aims to provide a high-speed modular router that can process millions of packets per second by arranging custom functions in a graph-like fashion. The advantage of FastClick is the possibility to re-arrange its rich set of internal elements to realize different types of packet processing applications. VPP should be considered when a fully-featured software network function (e.g., switch, router, or security appliance) is required. Its code was part of Cisco high-end routers before being released as open-source and therefore contains a large set of software components that can be used for all kinds of possible L2-L4 applications. VALE fulfills the role of a high-speed L2 learning switch that interconnects multiple VMs. Its main purpose is to provide a high-speed virtual local Ethernet switch.

## 4 TEST METHODOLOGY

To evaluate and compare software switch performance and to identify potential bottlenecks, we define four archetypal test scenarios,




(a) p2p (b) p2v (c) v2v (d) loopback

**Figure 2: The four test scenarios we propose. Red arrows illustrate the packet flow in the System Under Test (SUT), namely the software switches.**

p2p, p2v, v2v and loopback, and two performance metrics particularly relevant for NFV, throughput and latency. The four scenarios are illustrated in Fig. 2. Note that we assume a server with 2 dual-port NICs and denote software switches as System Under Test (SUT). This section describes each scenario. Some further configuration details are provided in the appendix.

**p2p (physical-to-physical)**. In this scenario, packets entering from one physical interface are forwarded to the other by the SUT, as shown in Fig. 2a. Although this scenario does not deal with VNFs, it is still relevant since common network functions are increasingly hosted by software switches, either to augment the physical NIC [38] or to reduce duplicated VNF processing [36]. Evaluating the bare forwarding rate between two physical interfaces thus provides a useful baseline reference. Furthermore, combined with other scenarios such as p2v and v2v, p2p helps to evaluate the overhead imposed by a virtualized environment, both qualitatively and quantitatively.

**p2v (physical-to-virtual)**. In this scenario, the SUT forwards packets between a physical interface and a VNF hosted in a virtualized environment, as shown in Fig. 2b. This scenario can be mapped to the first and last hop of NFV service function chains inside a server. Combined with p2p, p2v reveals SUT efficiency when connected to a virtualized environment.

**v2v (virtual-to-virtual)**. In this scenario, the SUT steers traffic between two VNFs, as shown in Fig. 2c. It helps to characterize how efficiently software switches perform such tasks. This is relevant for NFV as services are usually deployed in chains, requiring intensive traffic exchange between VNFs. It is important to note that v2v throughput is upper-bounded by memory bandwidth, while other scenarios that include physical interfaces are limited by NIC capacity. It is thus important to know how fast an SUT can forward traffic across VMs.

**loopback**: This scenario is representative of a complete NFV service chain. Packets entering from one NIC are steered by the SUT through a chain of VNFs before exiting through the other NIC. Each VNF is hosted by an independent VM. Fig. 2d shows the case of a 1-VNF service chain. We also take into account multi-VNF chains in our study.

We believe these four scenarios and two performance metrics are the most relevant for NFV. Their combination enables a comprehensive understanding of the performance bottlenecks of any software switch built for NFV purposes.

**Table 2: Software Switches Parameter Tuning**

| Solutions | Applied Tunings |
|---|---|
| FastClick | Increase descriptor ring size to 4096 |
| t4p4s | Remove source MAC learning phase |
| VALE | Disable flow control for NIC interfaces |

## 5 EXPERIMENTAL RESULTS

In this section, we first describe the testbed over which we execute our experiments and then present the results obtained by applying our test methodology to the seven software switches identified in Sec.2 and analyzed in Sec.3.

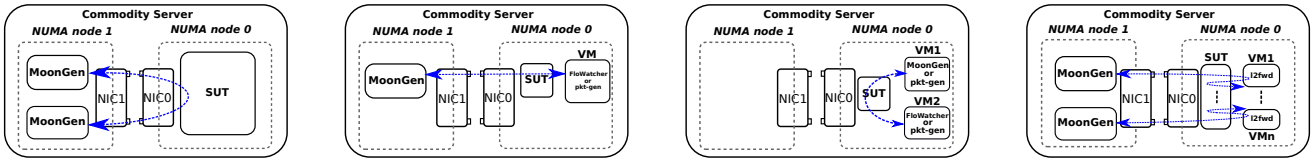
### 5.1 Measurement platform

Our testbed includes a commodity server equipped with two Intel Xeon E5-2690 v3 @ 2.60GHz CPUs (each with 24 virtual cores under Hyperthreading and 32k/256k/30720K L1-3 caches), and two Intel 82599ES dual-port 10-Gbps NICs spread over two NUMA nodes. The server runs Ubuntu 16.04.1 with Linux 4.8.0-41-generic kernel. VNFs are deployed inside CentOS 7 [1] VMs using QEMU 2.5.0.[2]

For each tested software switch, the latest stable version at the time of writing has been used, namely: FastClick (commit `8c9352e`); BESS (Haswell tarball); OvS-DPDK (version 2.11.90); Snabb (commit `771b55c`); VALE (commit `1b5361d`); t4p4s (commit `b1161b2`); and VPP (version 19.04). Moreover, we tune the default settings of some switches in order to optimize their performance while imposing minimal configuration. The adopted tuning is described in Table 2. Furthermore, as recommended in [12] and [17], we set the CPU frequency scaling governor to *"performance"* and disable Turbo-boost in order to reduce performance variance. We also reserve 1GB Hugepages to minimize TLB misses. Finally, some cores are deliberately isolated from the kernel using `isolcpus` and reserved solely for the software switch under test.

The setup for each test scenario is illustrated in Fig. 3. Software switches are always deployed on a single core on NUMA node 0 to ensure a fair comparison. Single-core is also arguably a reasonable assumption as network operators usually seek to limit resources devoted to networking. Each VM is allocated with four cores through the QEMU `-smp` option. By default, we use MoonGen (commit `31af6e6`) as traffic generator/receiver (TX/RX) for p2p and loopback scenarios. For p2v and v2v tests, all switches use MoonGen [29] and FloWatcher-DPDK [61], except VALE, which requires pkt-gen [7] as TX/RX inside the VM(s). This is because the VM's

[2]Newer versions of QEMU cannot be used due to their compatibility issues with BESS [37].




(a) p2p (b) p2v (c) v2v (d) loopback

**Figure 3: Test scenarios mapped to our testbed with two NUMA nodes each associated with a dual port 10Gbps NIC directly connected to the other NUMA node's NIC. Blue arrows represent the data flow.**

ptnet driver is tightly coupled with host VALE ports and can only render optimal performance with netmap compatible tools. Other switches do not have the same problem since the intermediate virtio rings decouple the guest VMs from the host.

It is important to note that the use of the same server for both traffic generation/reception and the system under test does not introduce spurious interference since the cores and memory are effectively isolated under the NUMA architecture of our server. In particular, we combine software switch utilities (e.g. DPDK EAL parameters) with system tools (e.g. `numactl`, `taskset`) to guarantee core and memory affinities. For the v2v scenario, everything runs on NUMA node 0 without the involvement of physical NICs, thus the traffic forwarding rate is only limited by the local memory speed. For other scenarios, the TX/RX components run on NUMA node 1 while the software switch under test (and TX/RX for p2v scenario) is deployed on NUMA node 0. The cores only access memory in their local NUMA node and do not share remote memory. Note that since packets are transferred through physical NICs, their maximum bandwidth (10Gbps) constitutes the theoretical bottleneck for these scenarios.

## 5.2 Throughput

In this section, we present and discuss the throughput sustained by the software switches in the four test scenarios under synthetic unidirectional/bidirectional traffic, with different packet sizes, namely 64, 256, 1024 Bytes.

*p2p scenario.* For this scenario we configure MoonGen to transmit synthetic traffic at 10 Gbps from NUMA node 1 to the SUT, as illustrated in Fig. 3a. Packets are sent at maximum rate disregarding any drops.[3] We vary the packet size (64B, 256B and 1024B) and measure the throughput (in Gbps) on NUMA node 1 by collecting outbound traffic from NUMA node 0.

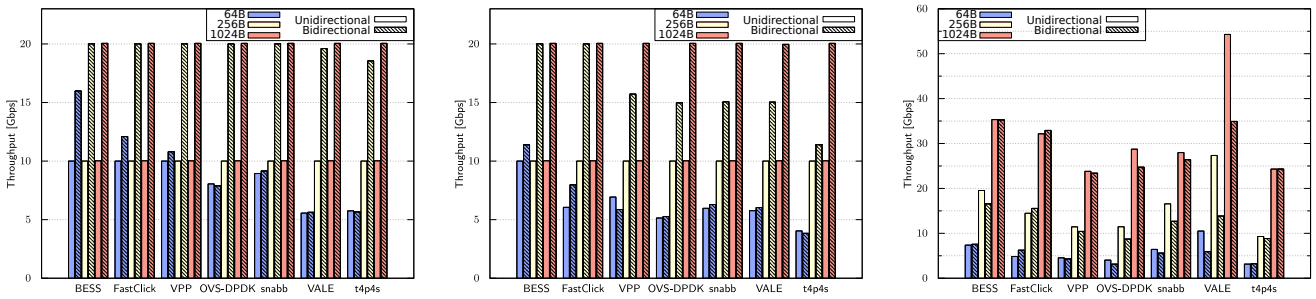
Fig. 4a shows the throughput results for the p2p scenario. Considering unidirectional traffic, all the software switches manage to saturate the 10 Gbps link with packets bigger than 256B, proving that they are all capable of handling realistic traffic (e.g., 850B average packet size in data centers [22]). For the most stressful input load with 64B packets, BESS, FastClick, and VPP still saturate the link at 10 Gbps (about 14.88 Mpps-million packets per second). Snabb achieves only 8.9 Gbps, as staging packets in internal buffers imposes extra overhead. OvS-DPDK achieves 8.05 Gbps due to the overhead imposed by its match/action pipeline. As the synthetic traffic consists of identical packets, corresponding to a single flow, OvS-DPDK's flow cache does not help. VALE switch and t4p4s present the worst performance of around 5.6 Gbps. VALE switch prioritizes memory isolation by design and therefore performs expensive packet copying operations between its ports, in addition to source MAC learning and flow table lookup. t4p4s incurs the overhead of implementing multiple stages, including header parsing/de-parsing and flow table lookup.

Bidirectional traffic is more stressful and provides a more relevant comparison of the throughput performance of BESS, VPP, and FastClick. This test is realized by simultaneously transmitting packets towards both interfaces of NUMA node 0 and measuring the aggregated throughput. As shown in Fig. 4a, all the switches, except VALE and t4p4s, reach 20 Gbps with 256B and 1024B packets. Note that such limited performance could be explained with the same considerations on unidirectional traffic results. For small 64B packets, BESS, FastClick, and VPP manage to exceed 10 Gbps. BESS even reaches 16 Gbps since it only performs very simple tasks like collecting statistics. FastClick additionally extracts and updates packet header fields while VPP performs a number of verifications. The other switches achieve similar results as with unidirectional traffic, as their bottleneck is the processing being less efficient and/or made of more complex operations.

*p2v scenario.* For the p2v scenario, we instantiate a single VM using the QEMU hypervisor. As mentioned before, VALE uses netmap's ptnet driver for high-performance VM networking.[4] We therefore need to use a customized version of QEMU provided by netmap authors [11], as it supports ptnet virtual interfaces with VALE ports as their host backend [47]. We then configure ptnet support in the guest VM and use pkt-gen (in RX mode) as traffic monitor. pkt-gen is based on the netmap API and provides native speed processing with ptnet virtual ports. To test other switches, we use standard QEMU and configure virtio virtual interfaces with vhost-user as host backend. Finally, we boost packet processing inside the guest VM by deploying DPDK (version 18.11) to bypass the guest kernel. We attach virtio interfaces to the DPDK `igb_uio` device driver and use FloWatcher-DPDK [61], a lightweight software traffic monitor, to measure the unidirectional throughput. Similar to pkt-gen, FloWatcher-DPDK performs measurement with negligible overhead [60] so that our results are not unduly influenced by this configuration discrepancy. The scenario is illustrated in Fig. 3b.

[3]Note that this is different from the usual Non-Drop-Rate (NDR) of RFC 2544 [26]: a binary search for the NDR is not suited for evaluating software solutions as it may converge to unreliable points due to even a single packet drop caused at the driver level.

[4]ptnet also supports passthrough of physical interfaces directly, without connecting them using VALE, but we decided not to use this feature because our work focuses on software switches and not on passthrough.




**(a) Throughput for the p2p scenario** **(b) Throughput for the p2v scenario** **(c) Throughput for the v2v scenario**

**Figure 4: Throughput in physical-to-physical (p2p), physical-to-virtual (p2v) and virtual-to-virtual (v2v) scenarios**

Results for the p2v scenario are shown in Fig. 4b. With unidirectional traffic, the software switches considered in our evaluation sustain 10 Gbps under 256B and 1024B packets. For 64B packets, FastClick, VPP, OvS-DPDK, and Snabb achieve only 5 to 7 Gbps, which is less than their p2p results. Thus, these switches experience a bottleneck in dealing with the virtualized environments as vhost-user requires to enqueue/dequeue virtio rings by copying packets. t4p4s achieves only 4.04 Gbps due to its less efficient processing pipeline and to the overhead imposed by vhost-user. On the other hand, VALE achieves a slightly higher throughput (5.77 Gbps) than in p2p (5.56 Gbps) since ptnet supports zero-copy packet delivery and imposes less overhead than vhost-user. Note that BESS sustains 10 Gbps regardless of the overhead from vhost-user, as the tasks it performs are very simple. The impact of vhost-user on BESS can only be understood with bidirectional traffic.

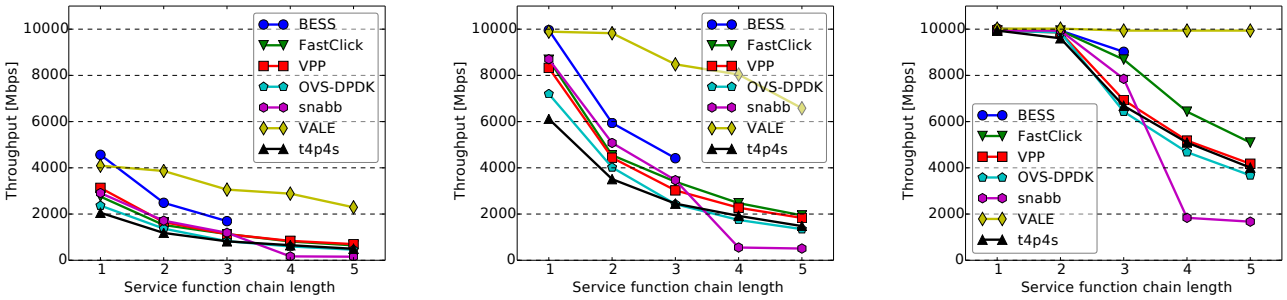
To obtain bidirectional traffic, we initiate two pkt-gen instances (for VALE)/one MoonGen instance (for others) to TX/RX from inside the VM, and start another MoonGen instance to TX/RX simultaneously on NUMA node 1, as illustrated in Fig.3b. However, we experienced severe performance degradation when the two pkt-gen instances are attached to the same ptnet port inside the VM. To overcome this, we attach the pkt-gen instances to a netmap virtual interface, which is in turn attached to the ptnet port through a VALE instance. Actually, this setting imposes an extra hop of packet forwarding, but this is the best we can do to have reasonable bidirectional p2v traffic with VALE. Without this issue, the bidirectional throughput of VALE is expected to be higher.

As shown in Fig. 4b, for bidirectional traffic with 256B and 1024B packets, BESS and FastClick are still able to sustain line rate, i.e., 20 Gbps, but the impact of vhost-user is noticeable for the other switches. Indeed, VPP, OvS-DPDK, Snabb, and t4p4s fail to saturate 20 Gbps, in contrast to their results in the bidirectional p2p test. VALE reaches only 15 Gbps due to the extra overhead imposed by the VALE instance inside the VM. The real throughput is expected to be higher than this and the results here only represent a lower bound. For 64B traffic, BESS achieves 11.38 Gbps, much lower than it achieved in the bidirectional p2p test (16 Gbps), further illustrating the impact of vhost-user. We also notice a throughput degradation for VPP (5.9 Gbps) compared to the unidirectional test (6.9 Gbps). To find out the cause, we reverse the unidirectional data path and transmit packets from the VM to NUMA node 1. This *"reversed"* unidirectional throughput is only 5.59 Gbps. It appears that VPP suffers from a performance penalty in receiving packets from vhost-user ports.

*v2v scenario.* For the v2v scenario, we instantiate two VMs, each with a virtual interface attached to the software switch under test. The virtual interface configuration is similar to the one for the p2v scenario. We deploy a traffic generator in the first VM and configure it to inject packets towards the software switch, which in turn forwards packets to the second VM running the monitoring VNF. Similarly to previous scenarios, different traffic monitoring/generation tools are required to realize the intended data path. For VALE unidirectional throughput, we deploy an instance of pkt-gen in each VM and configure them to perform traffic generation/monitoring respectively. For other switches, we run MoonGen on the first VM as traffic generator and FloWatcher-DPDK on the second VM to measure throughput. The v2v scenario configuration is illustrated in Fig 3c. Unlike other scenarios, v2v is only limited by the memory bandwidth and thus illustrates the upper limit of inter-VM communication using software switches. Benchmarking results confirmed that both MoonGen and pkt-gen are able to transmit packets at line rate in VMs.

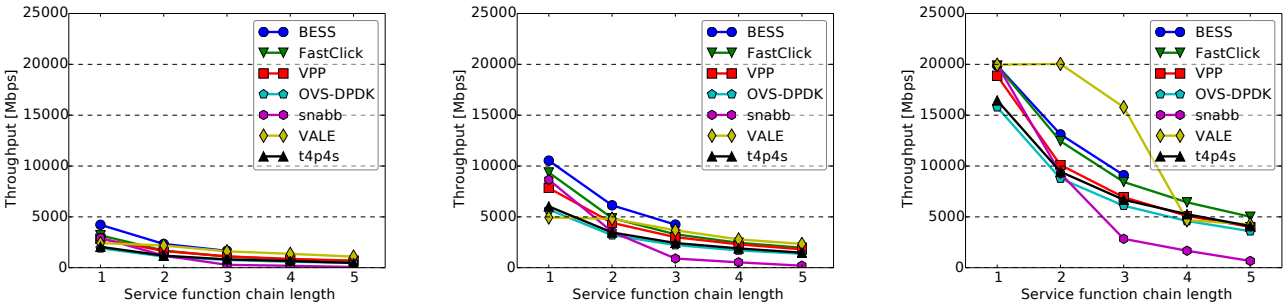
Results with unidirectional and bidirectional traffic are reported in Fig. 4c. With unidirectional traffic, VALE achieves 10.50 Gbps for 64B packets. Compared with its corresponding p2v result (5.77 Gbps), it is clear that VALE is more efficient in host-VM communication. This is mostly due to the efficiency of the ptnet zero-copy host-VM packet delivery mechanism. As shown in Fig. 4c, other switches achieve throughput lower than 7.4 Gbps. Snabb is the only one outperforming its p2v result (6.42 Gbps vs. 5.97 Gbps) while other switches experience throughput degradation. We argue that this is because of the different implementation details between Snabb and DPDK. With a larger packet size of such as 256B and 1024B, almost all solutions sustain line rate, i.e., 10 Gbps.

With bidirectional traffic, we deploy an instance of MoonGen in each VM as TX/RX for software switches other than VALE. For VALE, we deploy two pkt-gen instances in each VM and make them TX/RX simultaneously. Similar to the p2v bidirectional test, we attach both pkt-gen instances in each VM to a netmap virtual interface which is attached to the ptnet virtual interface through a VALE instance. As shown in Fig. 4c, switches exhibit lower forwarding rate with bidirectional traffic, compared to the experiments with




**(a) 64B packets** **(b) 256B packets** **(c) 1024B packets**

**Figure 5: Unidirectional throughput measurement of loopback scenario**




**(a) 64B packets** **(b) 256B packets** **(c) 1024B packets**

**Figure 6: Bidirectional throughput measurement of loopback scenario**

unidirectional traffic. For example, VALE achieves 35 Gbps with 1024B packets, which is only 64% of its unidirectional throughput. This occurs because bidirectional traffic doubles the number of packet copy operations between VALE ports and through virtio rings for the others.

*loopback scenario.* In the last scenario, loopback, we instantiate a chain of VMs, each of which is allocated four virtual cores and a pair of virtual interfaces. Each software switch steers traffic across the VMs in sequence, forming a service chain. Fig. 3d illustrates the setup. For VALE, we configure two ptnet virtual interfaces for each VM in which we run a VALE instance as a VNF. This VALE instance cross-connects the pair of ptnet ports. Each VM is linked to its successor through a VALE instance. The first and last VM also need to link the physical ports with two additional VALE instances. In all, we need $N + 1$ VALE instances for an $N$-VNF service chain. For the other switches, we configure two virtio interfaces with vhost-user backend for each VM, in which we run an instance of the DPDK l2fwd sample application [6] that cross-connects interfaces, updates the MAC addresses, and forwards packets in batches. On NUMA node 1, we start MoonGen to generate 10Gbps traffic through one port and measure throughput for different packet sizes from the other port. The SUT is configured to rely packets between MoonGen and the service chain. We vary the number of VNFs from 1 to 5 to test the throughput of each SUT with increasing service chain length.

Fig. 5 illustrates the throughput for unidirectional traffic. For the 1-VNF case, BESS yields the highest throughput. However, as we increase the service chain length with more VMs, it is outperformed by VALE. This is mainly due to the fact that BESS needs to perform an increasing number of packet copies with an increasing number of VMs.[5] Even though VALE still needs to copy packets between its VALE ports, this overhead is compensated by the efficient VM network I/O of ptnet. As shown in Fig. 5c, VALE manages to sustain 10 Gbps for unidirectional traffic with 1024B packets, regardless of the service chain length. Other switches achieve lower throughput due to the overhead (mainly packet copies) imposed by vhost-user. Note that when the service chain length reaches 4, Snabb becomes overloaded and its throughput plummets, as the workload is too much to handle with a single core.

Fig. 6 presents the results with bidirectional traffic. In this case, VALE performance significantly degrades, especially for smaller packets. For 1024B packets, its performance begins to drop when the service chain length is greater than 2. This is due to the dominant impact of doubling the overhead of packet copying between VALE

[5]Note that BESS exhibits QEMU compatibility issues that prevent the instantiation of more than 3 VMs simultaneously. As a result, we cannot obtain throughput or latency results for loopback scenario with more than 3 VNFs.

**Table 3: RTT latency ($\mu$s) for p2p scenario and loopback scenario with 1-4 VNFs.**

| Scenario | p2p | | | 1-VNF loopback | | | 2-VNF loopback | | | 3-VNF loopback | | | 4-VNF loopback | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $0.1R^+$ | $0.50R^+$ | $0.99R^+$ | $0.1R^+$ | $0.50R^+$ | $0.99R^+$ | $0.1R^+$ | $0.50R^+$ | $0.99R^+$ | $0.1R^+$ | $0.50R^+$ | $0.99R^+$ | $0.1R^+$ | $0.50R^+$ | $0.99R^+$ |
| BESS | 4.0 | 4.6 | 6.4 | 35 | 15 | 39 | 67 | 33 | 136 | 167 | 55 | 147 | - | - | - |
| FastClick | 5.3 | 7.8 | 8.4 | 69 | 26 | 37 | 164 | 47 | 70 | 368 | 73 | 129 | 978 | 107 | 149 |
| OvS-DPDK | 4.3 | 5.2 | 9.6 | 50 | 23 | 514 | 124 | 42 | 909 | 182 | 90 | 1052 | 235 | 124 | 336 |
| Snabb | 7.3 | 11.3 | 22 | 70 | 27 | 74 | 123 | 53 | 146 | 186 | 95 | 266 | 406 | 365 | 1181 |
| VPP | 4.5 | 5.9 | 13.1 | 41 | 20 | 47 | 116 | 47 | 74 | 175 | 73 | 98 | 231 | 87 | 131 |
| VALE | 32 | 34 | 59 | 32 | 35 | 65 | 41 | 51 | 90 | 54 | 74 | 132 | 67 | 100 | 166 |
| t4p4s | 32 | 31 | 174 | 169 | 65 | 2259 | 274 | 117 | 3911 | 434 | 192 | 5535 | 548 | 228 | 7275 |

ports. Other software switches also present decreasing throughput as the service chain length grows, due to the increasing number of memory copies between the SUT and VMs.

## 5.3 Latency

It is critical for network operators to understand the impact on packet latency of different switch architectures and design choices. In this section, we report measurements of round-trip time (RTT) latency, which in our case is defined as the time spent between packet emission by the traffic generator and the time the packet is received by the traffic monitor.

In order to perform meaningful measurements, it is necessary to identify the Maximal Forwarding Rate ($R^+$), defined as the maximal rate the SUT can forward packets without loss. Injecting packets at a speed higher than $R^+$ causes data path congestion and leads to packet losses that bias the measured latency. On the other hand, injecting packets at a very small rate may also impair latency as most solutions employ batch processing. It proved to be very hard to determine $R^+$ since software traffic generators generally lack the stability of hardware and may induce non-deterministic packet losses.[6] VNF chains in the loopback scenario tend to exacerbate this uncertainty. Rather than trying to identify the precise $R^+$, we follow the methodology introduced in [46] and define $R^+$ as the average throughput achieved under saturating input. We measure latency at loads of 0.10, 0.50, and 0.99 times $R^+$. Thus, $0.99R^+$ reflects the latency under heavy input load, $0.50R^+$ under intermediate, more normal load, while $0.10R^+$ shows the impact of batch processing on latency under low load.

We perform the described latency measurement specifically for p2p and loopback scenarios as, in these two scenarios, MoonGen can leverage the NIC to accurately and efficiently time-stamp UDP packets [29]. We have not performed a latency test for p2v as its RTT is expected to be similar to that of the loopback scenario with one VNF. For v2v, we cannot perform the same test as for p2p and loopback since virtual interfaces, unlike physical ones, do not support hardware time-stamping. Fortunately, MoonGen implements a software time-stamping feature that can still be utilized in VMs. Although less accurate than hardware time-stamping, it provides a means to compare different SUTs under the same setup.

*p2p scenario.* To measure RTT in the p2p scenario, MoonGen is configured with two threads. One thread generates synthetic traffic with 64B packets, as for measuring throughput. The other TX thread periodically injects into background traffic Precise Time Protocol (PTP) packets with specific sequence numbers, collects these special PTP packets on their way back from the other port of the NIC in NUMA node 1, and calculates the round-trip time based on the difference between TX and RX time-stamps. These time-stamps are generated by the underlying Intel 82599 NIC, under the instruction of MoonGen.

We list the measured average latency in $\mu$s in Table 3. Under $0.99R^+$ load, t4p4s presents a very high latency of 174 $\mu$s, showing its instability under peak load. Since the other DPDK solutions do not face such problems, we believe this is due to the inefficiency of the t4p4s internal pipeline. The hardware abstraction layer of t4p4s presents a trade-off between performance and platform independence and the level of abstraction can be re-factored to enhance performance. VALE also imposes high latency at high load as it uses interrupts for packet I/O unlike BESS, FastClick, OvS-DPDK, and VPP that exploit DPDK busy-waiting for packet I/O. Snabb latency is also quite high (22 $\mu$s), mainly because its LuaJIT compiler keeps evaluating its execution time in performing online code optimizations. Under $0.50R^+$ load, VALE presents the highest average latency of 34 $\mu$s, mainly because it uses interrupt instead of busy-waiting for packet I/O. t4p4s also presents a relatively high latency of 31 $\mu$s, due again to its inefficient internal pipeline. Among the other switches, Snabb achieves 11.3 $\mu$s, because of the extra delay imposed by intermediate inter-module buffers. Under $0.10R^+$ load, we observe that t4p4s achieves higher latencies than in the $0.50R^+$ test. This is a consequence of the delay in constituting batches under low load.

*loopback scenario.* The loopback latency test uses the same settings as the p2p test with $R^+$ set to the corresponding unidirectional loopback throughput. We list the measured average RTTs in $\mu$s with 1 to 4 VNFs in Table 3. For all the switches we consider, latency for $0.99R^+$ load is always higher than for $0.50R^+$ load. This is as expected, since $R^+$ is only the average throughput and the actual forwarding rate of each software switch fluctuates around it. Consequently, an unstable software switch might fail to sustain $0.99R^+$ in a specific time period, causing data path congestion and packet loss. Such a situation rarely happens under $0.50R^+$ load. Another important result is the impact of batch processing of VNFs since, at a low input rate, time has to be spent to wait for new packets to complete a batch, thus impairs overall latency. As shown in Table 3, latency under $0.10R^+$ load is higher than under $0.50R^+$ for all the software switches except VALE, mainly due to the strict batch processing of DPDK l2fwd. In particular, the ratio between 0.10 and $0.50R^+$ is more than 9 for FastClick with 4 VNFs, as FastClick also suffers from its own batch processing delay. VALE does not incur this issue since it dynamically adjusts the batch size. In most

[6] Precision is made more difficult by the low granularity of software traffic generators. MoonGen, for example, rounds up TX rates in the range [9.88,10] Gbps to 10 Gbps.

**Table 4: RTT latency ($\mu$s) for v2v scenario.**

| BESS | FastClick | OvS-DPDK | Snabb | VPP | VALE | t4p4s |
|---|---|---|---|---|---|---|
| 37 | 45 | 43 | 67 | 42 | 21 | 70 |

cases, t4p4s presents the worst latency. For $0.99R^+$ load, its latency is much higher than for other switches, reflecting the instability of its processing pipeline. Under the normal $0.50R^+$ load, its latency is at least two times higher, reflecting the inefficiency of its processing pipeline. OvS-DPDK is also unstable under high input load yielding very high latency. Note that in the 4-VNF scenario under $0.99R^+$ load, its latency ($336\mu$s) is smaller than for other scenarios ($\geq 514\mu$s). We conjecture the reason to be that the processing of VNFs becomes the dominating overhead as service chain length grows. Snabb presents a huge latency leap in the 4-VNF test since it is overloaded at this point and fails to keep up with input traffic. The same phenomenon was observed in its throughput test.

*v2v scenario.* Thanks to the good compatibility with the operating system, standard tools can be used to measure the latency for VALE in this scenario. We simply configure routing using `ip` command for each VM. We then ping the second VM from the first and get the average RTT. Other switches do not support system tools. Instead, as mentioned before, we measure latency using the software time-stamping feature of MoonGen since hardware time-stamping is not supported by virtual interfaces. The setup is similar to loopback tests: we configure two virtio interfaces for each VM. All the interfaces are attached to the SUT. In the first VM, we launch an instance of MoonGen with software time-stamping enabled. Packets are time-stamped and transmitted from one virtio interface towards the SUT, which forwards traffic to the second VM. The second VM in turn forwards the packets back to the SUT using the DPDK l2fwd application. Then the SUT sends the packets to the first VM. The MoonGen instance in the first VM time-stamps the received packets and calculates the RTT based on the difference between RX and TX time-stamps. Packets are transmitted at 672 Mbps (=1 Mpps) for all the tests. Although not as accurate as hardware time-stamping, this approach reveals the main characteristics of the solutions. Results are listed in Table 4. As for the v2v throughput test, VALE outperforms other switches with only 21 $\mu$s latency. BESS, FastClick, VPP, and OvS-DPDK realize very similar latencies (37-45 $\mu$s) as they all use vhost-user to interconnect VMs. While ptnet requires two packet copying operations between VALE ports, solutions based on vhost-user have to incur four copies on virtio rings. Snabb incurs a higher latency of 67 $\mu$s, because of the delay imposed by its internal ring buffers. t4p4s presents the worst latency of 70 $\mu$s due to its less efficient internal pipeline.

## 5.4 Software switch use cases

Our experience in conducting the present evaluation leads us to make the following remarks on possible use cases for the considered switches. These remarks complement the taxonomy previously presented in Table 1.

BESS achieves both high throughput and low latency in p2p, p2v, and 1-VNF loopback scenarios. It is a viable choice to switch traffic between physical NICs and one or multiple paralleled VMs. It has reasonable performance in 2 and 3-VNF loopback scenarios but suffers from its incompatibility with newer versions of QEMU hypervisors when the number of VMs increases (i.e., > 3).

Snabb performs well in most cases but suffers from overload in the loopback scenario with a chain of more than 3 VNFs. It is easier to deploy than other solutions based on DPDK or netmap and is thus a good choice when the time-to-production of specific applications is critical.

OVS-DPDK and t4p4s have the advantage of supporting OpenFlow and P4, respectively, and are thus the only solutions that work with third-party SDN controllers and newly introduced protocols. OVS-DPDK appears the best option for a stateless SDN scenario while t4p4s is preferable when some state is required (e.g., for a firewall).

FastClick and VPP have good performance in all scenarios and simplify VNF migration and redeployments thanks to the isolation provided by virtio. Moreover, unlike BESS, they are compatible with newer hypervisor versions and can therefore be used to build both linear and parallel NFV environments with reasonable trade-offs. Of these two solutions, VPP might be preferred when latency is the primary concern since it generally has lower RTT and avoids severe latency degradation at low input rates (e.g., $0.1R^+$).

Finally, VALE, augmented by ptnet pass-through, achieves relatively high throughput in v2v and loopback scenarios. It is therefore well-suited to construct linear service chains in environments with high workloads. On the other hand, as ptnet is highly dependent on the host netmap module, it does not have the same level of memory isolation as the virtio paravirtualized driver. Migrating its VNFs may therefore additionally require synchronization at host level. A further caveat is that VALE, as a simple Ethernet switch, has limited scope for classification compared to the other solutions and may require enhancement to support advanced traffic routing between VNFs.

# 6 CONCLUSION

The emergence of high-speed packet I/O frameworks and the proliferation of NFV have given rise to intense research on the design of software switches running on COTS platforms. Many different designs have been proposed and implemented to route traffic between NICs and VNFs in NFV applications. In this paper we have sought to improve understanding of the performance of these alternative designs by defining a performance measurement methodology and providing sample results for seven state-of-the-art proposals.

The methodology is based on four test scenarios, physical to physical (p2p), physical to virtual (p2v), virtual to virtual (v2v) and loopback (with multiple VNFs), designed to explore the performance of typical NFV configurations where traffic is steered between multiple physical and virtual interfaces. In the interest of reproducibility, the paper describes the experimental set-up in detail, including specifications of tested software and hardware versions and the packet generation and monitoring tools used. All the scripts and instructions used to run our experiments are available on GitHub [2].

The measurement results reveal that no one switch prevails in all scenarios. This is as expected given the different design objectives of the considered software switches but is a useful reminder that the best switch choice depends significantly on the intended

**Table 5: Software Switches Use Cases Summary**

| | Best at | Remarks |
|---|---|---|
| BESS | Forwarding between physical NICs | Incompatible with newer versions of QEMU |
| Snabb | Fast deployment, runtime optimization | Bottlenecked with multiple VNFs |
| OvS-DPDK | Stateless SDN deployments | Supports OpenFlow protocol |
| FastClick | VNF chaining | Supports live migration, high latency at low workload |
| VPP | VNF chaining | Supports live migration |
| VALE | VNF chaining with high workload | Limited traffic classification and live migration capability |
| t4p4s | Stateful SDN deployments | Supports P4 language |

NFV context. The presented results and related discussion enable a more informed choice and should guide the design of potential enhancements to relieve identified bottlenecks.

Our planned future work will include consideration of multi-core solutions and the use of containers instead of VMs. It is worth noting that the present wide-ranging comparison has required considerable effort, both to understand the detail of the considered switches and to set up and conduct the experiments. We hope therefore that other researchers will be able to profit from our experience in further exploring the performance dimensions of existing and emerging software switches and in further refining the evaluation methodology.

## ACKNOWLEDGMENTS

This work has been carried out at LINCS (http://www.lincs.fr) and benefited from NewNet@Paris, Cisco's Chair "Networks for the Future" at Telecom ParisTech (https://newnet.telecom-paristech.fr). We highly appreciate the support from them. We also would like to thank the Shepherd and the anonymous reviewers for their valuable comments that helped to improve the quality of the paper.

## A SWITCH CONFIGURATIONS

To enhance reproducibility, we provide the most critical configuration details for each test scenario. More detailed descriptions can be found in our Github repository [2].

### A.1 p2p

Each switch requires a unique configuration to realize the p2p scenario. We only show the most critical configuration snippet for each design. For BESS, we composed a configuration script in which physical interfaces are hooked to the *bessd* daemon process with the built-in `PMDPort` module. Physical queues (input/output) of the hooked interfaces are instantiated using `QueueInc/QueueOut` modules. Packet forwarding is realized by linking different queues with the right arrow:

```
inport::PMDPort(port_id=0,...)
outport::PMDPort(port_id=1,...)

in0::QueueInc(port=inport, qid=0)
out0::QueueOut(port=outport, qid=0)

in0 -> out0
```

For FastClick, we compose a similar configuration file with `FromDPDKDevice/ToDPDKDevice` modules that hook and link two physical interfaces as follows:

```
FromDPDKDevice(0,...)->ToDPDKDevice(1,...)
```

Note that it is easier just to whitelist the required physical interfaces using DPDK EAL "-w" option.

For t4p4s, we select its l2fwd application which forwards packets according to a predefined flow table. The table is configured with "destination MAC address/output port" as Match/Action fields. Traffic generators need to send packets with the corresponding destination MAC addresses to realize the required forwarding.

For VPP, we specify the PCI addresses of the interfaces in the configuration file. We interconnect the ports with the *l2patch* function, as this is functionally equivalent to the configuration of other switches:

```
test l2patch rx port0 tx port1
test l2patch rx port1 tx port0
```

For Snabb switch we similarly write a custom module that hooks the ports by PCI addresses and recompile the Snabb software to make the module executable. Inside the module, we start a new configuration object and instantiate two logical port "apps" using the PCI port addresses which are then interconnected through the "link" method:

```
local c = config.new()
config.app(c, "nic1",...,{pciaddr = pci1})
config.app(c, "nic2",...,{pciaddr = pci2})
config.link(c, "nic1.tx -> nic2.rx")
```

For OvS-DPDK, we configure a new bridge and attach the physical interfaces to it by specifying their PCI addresses using the `ovs-vsctl` command. Then we populate the flow table with direct forwarding rules between the interfaces using the `ovs-ofctl` command.

For netmap, we need to unload the `ixgbe` kernel module and load its netmap counterpart. The physical ports are thus bound to the netmap device driver. Then we simply bind physical ports to a VALE instance (in this case *vale0*) using the `vale-ctl` command:

```
vale-ctl -a vale0:p1
vale-ctl -a vale0:p2
```

### A.2 p2v

As for p2p, we need to follow switch specific approaches. The only difference is that we have to consider the virtual interface connecting software switches to VNFs. To interact with virtualized environments such as virtual machines or containers, each switch must create a virtual interface. Snabb, VPP, OvS-DPDK, FastClick and BESS achieve this using the vhost-user protocol. Netmap solutions, on the other hand, achieve this using *ptnet* [47]. Some

configurations are required on the VNF side to realize p2v workflow. These are described in Sec. 5. Here we specifically detail the minimal configuration required for each software switch. In particular, for BESS we configure a virtual interface "v1" using the `PMDPort` module by specifying the name and Unix domain socket path. Then physical interface "inport" is linked to "v1" to realize p2v workflow:

```
inport::PMDPort(port_id=0, ...)
in0::QueueInc(port=inport, qid=0)

v1::PMDPort(vdev="name,iface=path,...")

in0 -> PortOut(port=v1.name)
```

Similarly, for FastClick, t4p4s, and VPP, we create a virtual interface by specifying name and socket path through the DPDK "–vdev" option. Note that, by default, t4p4s does not work with virtual interfaces. We therefore disabled some offloading features and recompiled the source code to make it compatible with vhost-user. OvS-DPDK accomplishes the same by setting the type of virtual interface to dpdkvhostuser. The created interfaces behave just like physical ones and they can be linked to render the intended traffic steering behavior.

Unlike solutions based on DPDK, Snabb implements its own version of vhost-user backend. Consequently, we create a virtual interface "vi1" leveraging its customized "vhostuser" module:

```
config.app(c,"vi1",vhostuser.VhostUser,...)
```

As for netmap, we just create a virtual interface using `vale-ctl` and attach it to a VALE instance which relies traffic from the physical interface to the VNF:

```
vale-ctl -n v0
vale-ctl -a vale0:v0
```

### A.3 v2v

To configure software switches realizing v2v workflow, we simply instantiate two virtual interfaces and interconnect them as described in the p2v scenario.

### A.4 loopback

For loopback, physical and virtual interfaces are interconnected as described for p2p/p2v scenarios. Note that for t4p4s in the loopback scenario, the VNFs need to modify the destination MAC address of each traversing packet to realize the required forwarding behavior.

## REFERENCES


[1] 2019. CentOS.org. https://cloud.centos.org/centos/7/images/. (2019).
[2] 2019. Comparing the performance of software switches. https://github.com/ztz1989/software-switches/tree/artifacts. (2019).
[3] 2019. CSIT. https://wiki.fd.io/view/CSIT. (2019).
[4] 2019. CSIT-1904. https://docs.fd.io/csit/rls1904/report/. (2019).
[5] 2019. Home - DPDK. https://www.dpdk.org/. (2019).
[6] 2019. L2 Forwarding Sample Application (in Real and Virtualized Environments). https://doc.dpdk.org/guides-18.08/sample_app_ug/l2_forward_real_virtual.html. (2019).
[7] 2019. netmap/apps/pkt-gen at master. https://github.com/luigirizzo/netmap/tree/master/apps/pkt-gen. (2019).
[8] 2019. Open vSwitch. https://www.openvswitch.org/. (2019).
[9] 2019. Open vSwitch with DPDK. http://docs.openvswitch.org/en/latest/intro/install/dpdk/. (2019).
[10] 2019. PF_RING ZC (Zero Copy). https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/. (2019).
[11] 2019. QEMU fork adding netmap passthrough networking, e1000 paravirtualized adapter, and VMPI support. https://github.com/vmaffione/qemu. (2019).
[12] 2019. snabb: Maximizing deployment performance. https://github.com/snabbco/snabb/blob/master/src/doc/performance-tuning.md. (2019).
[13] 2019. Snabb NFV for OpenStack. https://github.com/snabbco/snabb-nfv. (2019).
[14] 2019. The LuaJIT Project. http://luajit.org/. (2019).
[15] 2019. Vhost-user Protocol. https://github.com/qemu/qemu/blob/master/docs/interop/vhost-user.txt. (2019).
[16] 2019. VPP - fd.io. https://wiki.fd.io/view/VPP. (2019).
[17] 2019. VPP/How To Optimize Performance (System Tuning). https://wiki.fd.io/view/VPP/How_To_Optimize_Performance_(System_Tuning). (2019).
[18] 2019. VSperf Home. https://wiki.opnfv.org/display/vsperf/VSperf+Home. (2019).
[19] Vamsi Addanki, Leonardo Linguaglossa, James Roberts, and Dario Rossi. 2019. Controlling software router resource sharing by fair packet dropping. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 1–9.
[20] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi. 2018. High-Speed Software Data Plane via Vectorized Packet Processing. *IEEE Communications Magazine* 56, 12 (December 2018), 97–103. https://doi.org/10.1109/MCOM.2018.1800069
[21] Tom Barbette, Cyril Soldani, and Laurent Mathy. 2015. Fast userspace packet processing. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 5–16.
[22] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2009. Understanding data center traffic characteristics. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 65–72.
[23] Nicola Bonelli, Andrea Di Pietro, Stefano Giordano, and Gregorio Procissi. 2012. On multi–gigabit packet capturing with multi–core commodity hardware. In *International Conference on Passive and Active Network Measurement*. Springer, 64–73.
[24] Nicola Bonelli, Gregorio Procissi, Davide Sanvito, and Roberto Bifulco. 2017. The acceleration of OfSoftSwitch. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 1–6.
[25] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.
[26] Scott Bradner and Jim McQuaid. 1999. RFC 2544 – Benchmarking methodology for network interconnect devices *(Internet Engineering Task Force (IETF))*.
[27] Sean Choi, Xiang Long, Muhammad Shahbaz, Skip Booth, Andy Keep, John Marshall, and Changhoon Kim. 2017. Pvpp: A programmable vector packet processor. In *Proceedings of the Symposium on SDN Research*. ACM, 197–198.
[28] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. 2009. RouteBricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 15–28.
[29] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference*. ACM, 275–287.
[30] Paul Emmerich, Daniel Raumer, Sebastian Gallenmüller, Florian Wohlfart, and Georg Carle. 2018. Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis. *Journal of Network and Systems Management* 26, 2 (2018), 314–338.
[31] Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2014. Performance characteristics of virtual switching. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. IEEE, 120–125.
[32] Vivian Fang, T Lvai, Sangjin Han, Sylvia Ratnasamy, Barath Raghavan, and Justine Sherry. 2018. Evaluating Software Switches: Hard or Hopeless? *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2018-136* (2018).
[33] Daniel Firestone. 2017. {VFP}: A Virtual Switch Platform for Host {SDN} in the Public Cloud. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 315–328.
[34] Sebastian Gallenmüller, Paul Emmerich, Rainer Schönberger, Daniel Raumer, and Georg Carle. 2017. Building Fast but Flexible Software Routers. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 101–102.
[35] Massimo Gallo and Rafael Laufer. 2018. Clicknf: a modular stack for custom network functions. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 745–757.
[36] Massimo Gallo and Fabio Pianese. 2018. vNS: A Modular Programmable Virtual Network Switch. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos (SIGCOMM '18)*. ACM, New York, NY, USA, 96–98. https://doi.org/10.1145/3234200.3234242
[37] Han. 2018. BESS connecting to VM. https://github.com/NetSys/bess/issues/874. (2018).

[38] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. 2015. SoftNIC: A software NIC to augment hardware. *Dept. EECS, Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2015-155* (2015).

[39] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. 2011. PacketShader: a GPU-accelerated software router. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 195–206.

[40] Michio Honda, Felipe Huici, Giuseppe Lettieri, and Luigi Rizzo. 2015. mSwitch: a highly-scalable, modular software switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 1.

[41] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. 2000. The Click modular router. *ACM Transactions on Computer Systems (TOCS)* 18, 3 (2000), 263–297.

[42] Sándor Laki, Dániel Horpácsi, Péter Vörös, Róbert Kitlei, Dániel Leskó, and Máté Tejfel. 2016. High speed packet forwarding compiled from protocol independent data plane specifications. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 629–630.

[43] Rafael Laufer, Massimo Gallo, Diego Perino, and Anandatirtha Nandugudi. 2016. Climb: Enabling network function composition with click middleboxes. *ACM SIGCOMM Computer Communication Review* 46, 4 (2016), 17–22.

[44] Giuseppe Lettieri, Vincenzo Maffione, and Luigi Rizzo. 2017. A Survey of Fast Packet I/O Technologies for Network Function Virtualization. In *International Conference on High Performance Computing*. Springer, 579–590.

[45] Leonardo Linguaglossa, Stanislav Lange, Salvatore Pontarelli, Gábor Rétvári, Dario Rossi, Thomas Zinner, Roberto Bifulco, Michael Jarschel, and Giuseppe Bianchi. 2019. Survey of Performance Acceleration Techniques for Network Function Virtualization [45pt]. *Proc. IEEE* (2019).

[46] Leonardo Linguaglossa, Dario Rossi, Salvatore Pontarelli, Dave Barach, Damjan Marjon, and Pierre Pfister. 2019. High-speed data plane and network functions virtualization by vectorizing packet processing. *Computer Networks* 149 (2019), 187 – 199. https://doi.org/10.1016/j.comnet.2018.11.033

[47] Vincenzo Maffione, Luigi Rizzo, and Giuseppe Lettieri. 2016. Flexible virtual machine networking using netmap passthrough. In *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 1–6.

[48] Zhixiong Niu, Hong Xu, Libin Liu, Yongqiang Tian, Peng Wang, and Zhenhua Li. 2017. Unveiling performance of NFV software dataplanes. In *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*. ACM, 13–18.

[49] Michele Paolino, Nikolay Nikolaev, Jeremy Fanguede, and Daniel Raho. 2015. SnabbSwitch user space virtual switch benchmark and performance optimization for NFV. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, 86–92.

[50] Nikolai Pitaev, Matthias Falkner, Aris Leivadeas, and Ioannis Lambadaris. 2018. Characterizing the performance of concurrent virtualized network functions with OVS-DPDK, FD. IO VPP and SR-IOV. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. ACM, 285–292.

[51] Nikolai Pitaev, Matthias Falkner, Aris Leivadeasy, and Ioannis Lambadarisy. 2017. Multi-VNF performance characterization for virtualized network functions. In *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 1–5.

[52] Reza Rahimi, Malathi Veeraraghavan, Yoshihiro Nakajima, Hirokazu Takahashi, S Okamoto, and N Yamanaka. 2016. A high-performance OpenFlow software switch. In *2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 93–99.

[53] Kaushik Kumar Ram, Alan L Cox, Mehul Chadha, and Scott Rixner. 2013. Hyper-switch: A scalable software virtual switching architecture. In *Presented as part of the 2013 {USENIX} Annual Technical Conference ({USENIX}{ATC} 13)*. 13–24.

[54] Luigi Rizzo. 2012. Netmap: a novel framework for fast packet I/O. In *21st USENIX Security Symposium (USENIX Security 12)*. 101–112.

[55] Luigi Rizzo and Giuseppe Lettieri. 2012. Vale, a switched ethernet for virtual machines. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 61–72.

[56] Rusty Russell. 2008. virtio: towards a de-facto standard for virtual I/O devices. *ACM SIGOPS Operating Systems Review* 42, 5 (2008), 95–103.

[57] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. 2016. Pisces: A programmable, protocol-independent software switch. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 525–538.

[58] Sivasothy Shanmugalingam, Adlen Ksentini, and Philippe Bertin. 2016. DPDK Open vSwitch performance validation with mirroring feature. In *2016 23rd International Conference on Telecommunications (ICT)*. IEEE, 1–6.

[59] Maryam Tahhan, Billy O'Mahony, and Al Morton. 2017. Benchmarking Virtual Switches in the Open Platform for NFV (OPNFV). RFC 8204. (Sept. 2017). https://doi.org/10.17487/RFC8204

[60] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, and Dario Rossi. 2018. FlowMon-DPDK: Parsimonious per-flow software monitoring at line rate. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*. IFIP/IEEE, 1–8.

[61] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, and Dario Rossi. 2019. FloWatcher-DPDK: lightweight line-rate flow-level monitoring in software. *IEEE Transactions on Network and Service Management* (2019).

[62] Tianzhu Zhang, Leonardo Linguaglossa, James Roberts, Luigi Iannone, Massimo Gallo, and Paolo Giaccone. 2019. A benchmarking methodology for evaluating software switch performance for NFV. In *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 251–253.

[63] Dong Zhou, Bin Fan, Hyeontaek Lim, Michael Kaminsky, and David G Andersen. 2013. Scalable, high performance ethernet forwarding with cuckooswitch. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 97–108.