Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis

Paul Emmerich · Daniel Raumer · Sebastian Gallenmüller · Florian Wohlfart · Georg Carle

Abstract Virtual switches, like Open vSwitch, have emerged as an important part of today's data centers. They connect interfaces of virtual machines and provide an uplink to the physical network via network interface cards. We discuss usage scenarios for virtual switches involving physical and virtual network interfaces. We present extensive black-box tests to quantify the throughput and latency of software switches with emphasis on the market leader, Open vSwitch. Finally, we explain the observed effects using white-box measurements.

Keywords Network measurement · Cloud · Performance evaluation · Performance characterization · MoonGen

1 Introduction

Software switches form an integral part of any virtualized computing setup. They provide network access for virtual machines (VMs) by linking virtual and also physical network interfaces. The deployment of software switches in virtualized environments has led to the extended term *virtual switches* and paved the way for the mainstream adoption of software switches [37], which did not receive much attention before. In order to meet the requirements in a virtualized environment, new virtual switches have been developed that focus on performance and provide advanced features in addition to the traditional benefits of software switches: high flexibility, vendor independence, low costs, and conceptual benefits for switching without Ethernet bandwidth limitations. The most popular virtual switch implementation – *Open vSwitch* (OvS [43]) – is heavily used in cloud computing frameworks like OpenStack [7] and OpenNebula [6]. OvS is an open source project that is backed by an active community, and supports common standards such as OpenFlow, SNMP, and IPFIX.

The performance of packet processing in software depends on multiple factors including the underlying hardware and its configuration, the network stack of the operating system, the virtualization hypervisor, and traffic characteristics (e.g., packet size, number of flows). Each factor can significantly hurt the performance, which gives the motivation to perform systematic experiments to study the performance of virtual switching. We carry out experiments to quantify performance influencing factors and describe the overhead that is introduced by the network stack of virtual machines, using Open vSwitch in representative scenarios.

Knowing the performance characteristics of a switch is important when planning or optimizing the deployment of a virtualization infrastructure. We show how one can drastically improve performance by using a different IO-backend for Open vSwitch. Explicitly mapping virtual machines and interrupts to specific cores is also an important configuration of a system as we show with a measurement.

The remainder of this paper is structured as follows: Section 2 provides an overview of software switching. We explain recent developments in hardware and software that enable sufficient performance in general purpose PC systems based on commodity hardware, highlight challenges, and provide an overview of Open vSwitch. Furthermore, we present related work on performance measurements in Section 3. The following Sec-

P. Emmerich · D. Raumer · S. Gallenmüller · F. Wohlfart · G. Carle
Technical University of Munich, Department of Informatics, Chair of Network Architectures and Services
Boltzmannstr. 3, 85748 Garching, Germany
E-mail: {emmericp|raumer|gallenmu|wohlfart|carle}@net.in.tum.de

tion 4 explains the different test setups for the measurements of this paper. Section 5 and Section 6 describe our study on the performance of software switches and their delay respectively. Ultimately, Section 7 sums up our results and gives advice for the deployment of software switches.

2 Software Switches

A traditional hardware switch relies on special purpose hardware, e.g., content addressable memory to store the forwarding or flow table, to process and forward packets. In contrast, a software switch is the combination of commodity PC hardware and software for packet switching and manipulation. Packet switching in software grew in importance with the increasing deployment of host virtualization. Virtual machines (VMs) running on the same host system must be interconnected and connected to the physical network. If the focus lies on switching between virtual machines, software switches are often referred to as virtual switches. A virtual switch is an addressable switching unit of potentially many software and hardware switches spanning over one or more physical nodes (e.g., the ”One Big Switch” abstraction [29]). Compared to the default VM bridging solutions, software switches like OvS are more flexible and provide a whole range of additional features like advanced filter rules to implement firewalls and per-flow statistics tracking.

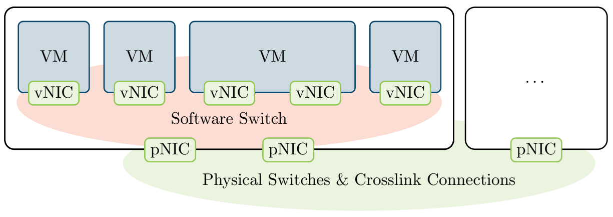


Fig. 1 Application scenario of a virtual switch

Figure 1 illustrates a typical application for virtual switching with both software and hardware switches. The software switch connects the virtual network interface cards (NIC) `vNIC` with the physical NICs `pNIC`. Typical applications in virtualization environments include traffic switching from pNIC to vNIC, vNIC to pNIC, and vNIC to vNIC. For example, OpenStack recommends multiple physical NICs to separate networks and forwards traffic between them on *network nodes* that implement firewalling or routing functionality [39]. As components of future network architectures packet flows traversing a chain of VMs are also discussed [33].

The performance of virtual data plane forwarding capabilities is a key issue for migrating existing services into VMs when moving from a traditional data center to a cloud system like OpenStack. This is especially important for applications like web services which make extensive use of the VM's networking capabilities.

Although hardware switches are currently the dominant way to interconnect physical machines, software switches like Open vSwitch come with a broad support of OpenFlow features and were the first to support new versions. Therefore, pNIC to pNIC switching allows software switches to be an attractive alternative to hardware switches in certain scenarios. For software switches the number of entries in the flow table is just a matter of configuration whereas it is limited to a few thousand in hardware switches [49].

2.1 State of the Art

Multiple changes in the system and CPU architectures significantly increase the packet processing performance of modern commodity hardware: integrated memory controllers in CPUs, efficient handling of interrupts, and offloading mechanisms implemented in the NICs. Important support mechanisms are built into the network adapters: checksum calculations and distribution of packets directly to the addressed VM [8]. NICs can transfer packets into memory (DMA) and even into the CPU caches (DCA) [4] without involving the CPU. DCA improves the performance by reducing the number of main memory accesses [26]. Further methods such as interrupt coalescence aim at allowing batch style processing of packets. These features mitigate the effects of interrupt storms and therefore reduce the number of context switches. Network cards support modern hardware architecture principles such as multi-core setups: Receive Side Scaling (RSS) distributes incoming packets among queues that are attached to individual CPU cores to maintain cache locality on each packet processing core.

These features are available in commodity hardware and the driver needs to support them. These considerations apply for packet switching in virtual host environments as well as between physical interfaces. As the CPU proves to be the main bottleneck [18, 35, 13, 47] features like RSS and offloading are important to reduce CPU load and help to distribute load among the available cores.

Packet forwarding apps such as Open vSwitch [43, 5], the Linux router, or Click Modular Router [31] avoid copying packets when forwarding between interfaces by performing the actual forwarding in a kernel module.

However, forwarding a packet to a user space application or a VM requires a copy operation with the standard Linux network stack. There are several techniques based on memory mapping that can avoid this by giving a user space application direct access to the memory used by the DMA transfer. Prominent examples of frameworks that implement this are PF_RING DNA [16], netmap [46], and DPDK [9, 1]. E.g., with DPDK running on an Intel Xeon E5645 (6x 2.4 GHz cores) an L3 forwarding performance of 35.2 Mpps can be achieved [9]. We showed in previous work that these frameworks not only improve the throughput but also reduce the delay [21]. Virtual switches like VALE [48] achieve over 17 Mpps vNIC to vNIC bridging performance by utilizing shared memory between VMs and the hypervisor. Prototypes similar to VALE exist [45, 33]. Virtual switches in combination with guest OSes like ClickOS [33] achieve notable performance of packet processing in VMs. All these techniques rely on changes made to drivers, VM environments, and network stacks. These modified drivers are only available for certain NICs. Experiments which combine OvS with the described high-speed packet processing frameworks [44, 47] demonstrate performance improvements.

2.2 Packet Reception in Linux

The packet reception mechanism implemented in Linux is called NAPI. Salim et al. [50] describe NAPI in detail. A network device signals incoming traffic to the OS by triggering interrupts. During phases of high network load, the interrupt handling can overload the OS. To keep the system reactive for tasks other than handling these interrupts, a NAPI enabled device allows reducing the interrupts generated. Under high load one interrupt signals the reception of multiple packets.

A second possibility to reduce the interrupts is offered by the Intel network driver. There the Interrupt Throttling Rate (ITR) specifies the maximum number of interrupts per second a network device is allowed to generate. The following measurements use the `ixgbe` driver, which was investigated by Beifuß et al. [11]. This driver has an ITR of 100,000 interrupts / second in place for traffic below 10 MB/s (156.25 kpps), the ITR is decreased to 20,000 if the traffic hits up to 20 MB/s (312.5 kpps), above that throughput value the ITR is reduced to 8,000.

2.3 Open vSwitch

Open vSwitch [5, 41, 42, 43] can be used both as a pure virtual switch in virtualized environments and as a general purpose software switch that connects physically separated nodes. It supports OpenFlow and provides advanced features for network virtualization.

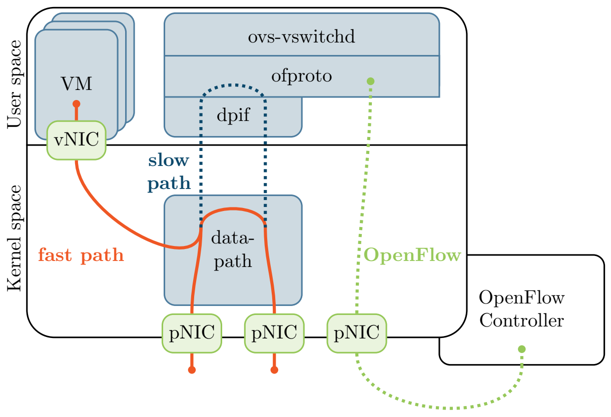


Fig. 2 Open vSwitch architecture representing the data processing flows

Figure 2 illustrates the different processing paths in OvS. The two most important components are the switch daemon `ovs-vswitchd` that controls the kernel module and implements the OpenFlow protocol, and the *datapath*, a kernel module that implements the actual packet forwarding. The datapath kernel module processes packets using a rule-based system: It keeps a flow table in memory, which associates flows with actions. An example for such a rule is forwarding all packets with a certain destination MAC address to a specific physical or virtual port. Rules can also filter packets by dropping them depending on specific destination or source IP addresses. The ruleset supported by OvS in its kernel module is simpler than the rules defined by OpenFlow. These simplified rules can be executed faster than the possibly more complex OpenFlow rules. So the design choice to explicitly not support all features OpenFlow offers results in a higher performance for the kernel module of OvS [42].

A packet that can be processed by a datapath-rule takes the *fast path* and is directly processed in the kernel module without invoking any other parts of OvS. Figure 2 highlights this fast path with a solid orange line. Packets that do not match a flow in the flow table are forced on the *slow path* (dotted blue line), which copies the packet to the user space and forwards it to the OvS daemon in the user space. This is similar to the encapsulate action in OpenFlow, which forwards a packet that cannot be processed directly on a switch to an OpenFlow controller. The slow path is implemented by the `ovs-vswitchd` daemon, which operates on OpenFlow rules. Packets that take this path are matched against OpenFlow rules, which can be added by an external OpenFlow controller or via a command line interface. The daemon derives datapath rules for packets based on the OpenFlow rules and installs them in the kernel module so that future packets of this flow can take the fast path. All rules in the datapath are associated with an inactivity timeout. The flow table in the datapath therefore only contains the required rules to handle the currently active flows, so it acts as a cache for the bigger and more complicated OpenFlow flow table in the slow path.

3 Related Work

Detailed performance analysis of PC-based packet processing systems have been continuously addressed in the past. In 2005, Tedesco et al. [51] presented measured latencies for packet processing in a PC and subdivided them to different internal processing steps. In 2007, Bolla and Bruschi [13] performed pNIC to pNIC measurements (according to RFC 2544 [14]) on a software router based on Linux 2.6[1]. Furthermore, they used profiling to explain their measurement results. Dobrescu et al. [18] revealed performance influences of multi-core PC systems under different workloads [17]. Contributions to the state of the art of latency measurements in software routers were also made by Angrisani et al. [10] and Larsen et al. [32] who performed a detailed analysis of TCP/IP traffic latency. However, they only analyzed the system under low load while we look at the behavior under increasing load up to 10 Gbit/s. A close investigation of the of latency in packet processing software like OvS is presented by Beifuß et al. [11].

In the context of different modifications to the guest and host OS network stack (cf. Section 2.1), virtual switching performance was measured [48, 33, 15, 44, 47, 27] but the presented data provide only limited possibility for direct comparison. Other studies addressed the performance of virtual switching within a performance analysis of cloud datacenters [53], but provide less detailed information on virtual switching performance.

Running network functions in VMs and connecting them via a virtual switch can be used to implement network function virtualization (NFV) with service function chaining (SFC) [23]. Martins et al. present ClickOS, a software platform for small and resource-efficient virtual machines implementing network functions [34]. Niu et al. discuss the performance of ClickOS [34] and SoftNIC [24] when used to implement SFC [38]. Panda et al. consider the overhead of virtualization too high to implement SFC and present NetBricks [40], a NFV framework for writing fast network functions in the memory-safe language Rust. Our work does not focus on NFV: we provide benchmark results for Open vSwitch, a mature and stable software switch that supports arbitrary virtual machines.

The first two papers from the OvS developers [41, 42] only provide coarse measurements of throughput performance in bits per second in vNIC to vNIC switching scenarios with Open vSwitch. Neither frame lengths nor measurement results in packets per second (pps) nor delay measurements are provided. In 2015 they published design considerations for efficient packet processing and how they reflect in the OvS architecture [43]. In this publication, they also presented a performance evaluation with focus on the FIB lookup, as this is supported by hierarchical caches in OvS. In [12] the authors measured a software OpenFlow implementation in the Linux kernel that is similar to OvS. They compared the performance of the data plane of the Linux `bridge-utils` software, the IP forwarding of the Linux kernel and the software implementation of OpenFlow and studied the influence of the size of the used lookup tables. A basic study on the influence of QoS treatment and network separation on OvS can be found in [25]. The authors of [28] measured the sojourn time of different OpenFlow switches. Although the main focus was on hardware switches, they measured a delay between 35 and 100 microseconds for the OvS datapath. Whiteaker et al. [54] observed a long tail distribution of latencies when packets are forwarded into a VM but their measurements were restricted to a 100 Mbit/s network due to hardware restrictions of their time stamping device. Rotsos et al. [49] presented OFLOPS, a framework for OpenFlow switch evaluation. They applied it, amongst others, to Open vSwitch. Deployed on systems with a NetFPGA the framework measures accurate time delay of OpenFlow table updates but not the data plane performance. Their study revealed actions that can be performed faster by software switches than by hardware switches, e.g., requesting statistics. We previously presented delay measurements of VM network packet processing in selected setups on an application level [21].

Latency measurements are sparse in the literature as they are hard to perform in a precise manner [20]. Publications often rely on either special-purpose hardware, often only capable of low rates, (e.g., [13, 54]) or on crude software measurement tools that are not precise enough to get insights into latency distributions on low-latency devices such as virtual switches. We use our packet generator MoonGen that supports hardware

[1] The "New API" network interface was introduced with this kernel version.

timestamping on Intel commodity NICs for the latency evaluation here [20].

We addressed the throughput of virtual switches in a previous publication [22] on which this paper is based. This extended version adds latency measurements and new throughput measurements on updated software versions of the virtual switches.

4 Test Setup

The description of our test setup reflects the specific hardware and software used for our measurements and includes the various VM setups investigated. Figure 3 shows the server setup.

4.1 Hardware Setup for Throughput Tests

Our device under test (*DuT*) is equipped with an Intel X520-SR2 and an Intel X540-T2 dual 10 GbE network interface card which are based on the Intel 82599 and Intel X540 Ethernet controller. The processor is a 3.3 GHz Intel Xeon E3-1230 V2 CPU. We disabled Hyper-Threading, Turbo Boost, and power saving features that scale the frequency with the CPU load because we observed measurement artifacts caused by these features.

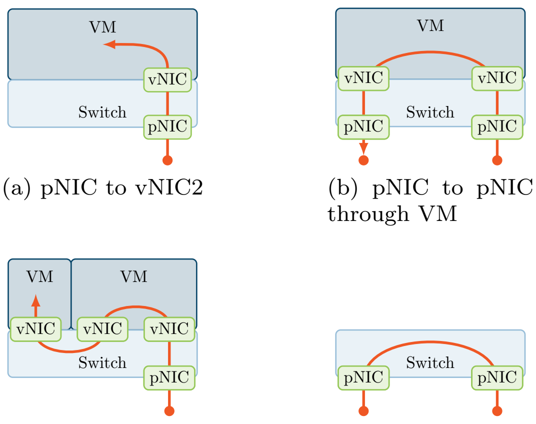
In black-box tests we avoid any overhead on the DuT through measurements, so we measure the offered load and the packet rate on the packet generator and sink. The DuT runs the Linux tool `perf` for white-box tests; this overhead reduces the maximum packet rate by $\sim 1\%$.

Figure 3 shows the setups for tests involving VMs on the DuT (Figure 3a, 3b, and 3c) and a pure pNIC switching setup (Figure 3d), which serves as baseline for comparison.

4.2 Software Setup

The DuT runs the Debian-based live Linux distribution Grml with a 3.7 kernel, the ixgbe 3.14.5 NIC driver with interrupts statically assigned to CPU cores, OvS 2.0.0 and OvS 2.4.0 with DPDK using a static set of OpenFlow rules, and qemu-kvm 1.1.2 with VirtIO network adapters unless mentioned otherwise.

The throughput measurements use the packet generator `pfsend` from the PF_RING DNA [16] framework. This tool is able to generate minimally sized UDP packets at line rate on 10 Gbit interfaces (14.88 Mpps). The packet rate is measured by utilizing statistics registers of the NICs on the packet sink.



(a) pNIC to vNIC2

(b) pNIC to pNIC through VM

(c) vNIC to vNIC

(d) pNIC forwarding

Fig. 3 Investigated test setups

As the PF_RING based packet generator does not support delay measurements, we use our traffic generator MoonGen [20] for those. This tool also can generate full line rate traffic with minimally sized UDP packets. MoonGen uses hardware timestamping features of Intel commodity NICs to measure latencies with sub-microsecond precision and accuracy. MoonGen also precisely controls the inter-departure times of the generated packets. The characteristics of the inter-packet spacings are of particular interest for latency measurements as it can have significant effects on the processing batch size in the system under test [20].

4.3 Setup with Virtual Machines

Figure 3a, 3b, and 3c show the setups for tests involving VMs on the DuT. Generating traffic efficiently directly inside a VM proved to be a challenging problem because both of our load generators are based on packet IO frameworks, which only work with certain NICs. Porting them to a virtualization-aware packet IO framework (e.g., vPF_RING [15]), would circumvent the VM-hypervisor barrier, which we are trying to measure.

The performance of other load generators was found to be insufficient, e.g., the `iperf` utility only managed to generate 0.1 Mpps. Therefore, we generate traffic externally and send it through a VM. A similar approach to load generation in VMs can be found in [2]. Running profiling in the VM shows that about half of the time is spent receiving traffic and half of it is spent sending traffic out. Therefore, we assume that the maximum possible packet rate for a scenario in which a VM internally generates traffic is twice the value we measured in the scenario where traffic is sent through a VM.

4.4 Adoptions for Delay Measurements

Precise and accurate latency measurements require a synchronized clock on the packet generator and sink. To avoid complex synchronization, we send the output from the DuT back to the source host. The measurement server generates traffic on one port, the DuT forwards traffic between the two ports of its NIC and sends it back to second port on the measurement server. Therefore, the delay measurements are not possible on all Setups (cf. Figure 3). For the first delay measurements the DuT forwards traffic between two pNICs as depicted in Figure 3d. We use these results as baseline to compare them with delays in the second setup, which uses a VM to forward traffic between the two pNICs as shown in Figure 3b.

Our latency measurements require the traffic to be sent back to the source. We used a X540 NIC for the latency tests because this interface was the only available dual port NIC in our testbed.

5 Throughput Measurements

We ran tests to quantify the throughput of several software switches with a focus on OvS in scenarios involving both physical and virtual network interfaces. Throughput can be measured as packet rate in Mpps or bandwidth in Gbit/s. We report the results of all experiments as packet rate at a given packet size.

5.1 Throughput Comparison

Table 1 compares the performance of several forwarding techniques with a single CPU core per VM and switch. DPDK vSwitch started as a port of OvS to the user space packet processing framework DPDK [3] and was later merged into OvS. DPDK support is a compile-time option in recent versions of OvS. We use the name DPDK vSwitch here to refer to OvS with DPDK support enabled.

DPDK vSwitch is the fastest forwarding technique, but it is still experimental and not yet ready for real-world use: we found it cumbersome to use and it was prone to crashes requiring a restart of all VMs to restore connectivity. Moreover, the administrator needs to statically assign CPU cores to DPDK vSwitch. It then runs a busy wait loop that fully utilizes these CPU cores at all times – there is 100% CPU load, even when there is no network load. There is no support for any power-saving mechanism or yielding the CPU between packets. This is a major concern for green computing,

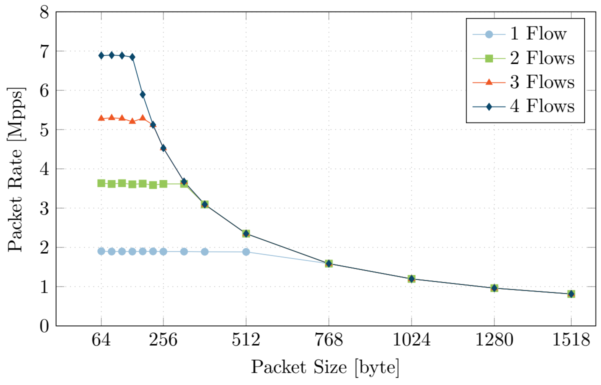


Fig. 4 Packet rate with various packet sizes and flows

efficiency, and resource allocation. Hence, our main focus is the widely deployed kernel forwarding techniques, but we also include measurements for DPDK vSwitch to show possible improvements for the next generation of virtual switches.

Guideline 1 Use Open vSwitch instead of the Linux bridge or router.

Open vSwitch proves to be the second fastest virtual switch and the fastest one that runs in the Linux kernel. The Linux bridge is slightly faster than IP forwarding when it is used as a virtual switch with vNICs. IP forwarding is faster when used between pNICs. This shows that OvS is a good general purpose software switch for all scenarios. The rest of this section will present more detailed measurements of OvS. All VMs were attached via VirtIO interfaces.

There are two different ways to include VMs in DPDK vSwitch: Intel *ivshmem* and *vhost user* with VirtIO. Intel *ivshmem* requires a patched version of qemu and is designed to target DPDK applications running inside the VM. The latter is a newer implementation of the VM interface and the default in DPDK vSwitch and works with stock qemu and targets VMs that are not running DPDK.

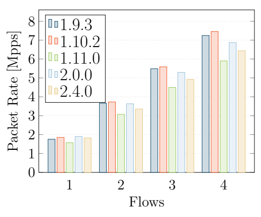
Intel *ivshmem* is significantly faster with DPDK running inside the VM [2]. However, it was removed from DPDK in 2016 [19] due to its design issues and low number of users [52]. The more generic, but slower, *vhost user* API connects VMs via the stable and standardized VirtIO interface [36]. All our measurements involving VMs in DPDK vSwitch were thus conducted with *vhost user* and VirtIO.

5.2 Open vSwitch Performance in pNIC to pNIC Forwarding

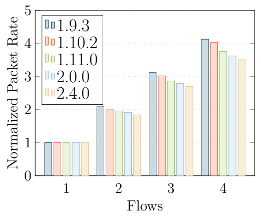
Figure 4 shows the basic performance characteristics of OvS in an unidirectional forwarding scenario between

Table 1 Single Core Data Plane Performance Comparison

	Mpps from pNIC to			
Application	pNIC	vNIC	vNIC to pNIC	vNIC to vNIC
Open vSwitch	1.88	0.85	0.3	0.27
IP forwarding	1.58	0.78	0.19	0.16
Linux bridge	1.11	0.74	0.2	0.19
DPDK vSwitch	13.51	2.45	1.1	1.0



(a) Packet rates



(b) Normalized to one flow

Fig. 5 Packet rate of different Open vSwitch versions, 1 to 4 flows

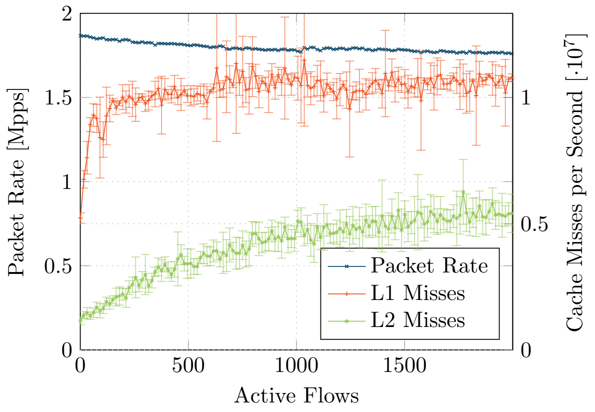


Fig. 6 Flow table entries vs. cache misses

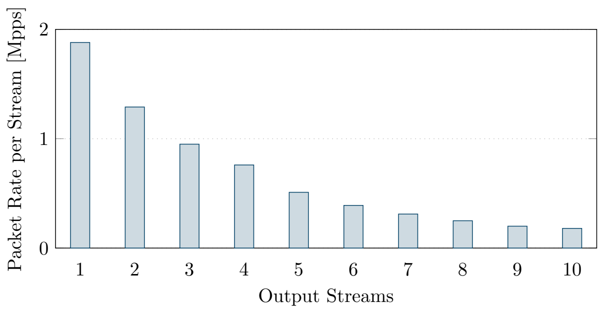


Fig. 7 Effects of cloning a flow

two pNICs with various packet sizes and flows. Flow refers to a combination of source and destination IP addresses and ports. The packet size is irrelevant until the bandwidth is limited by the 10 Gbit/s line rate. We ran further tests in which we incremented the packet size in steps of 1 Byte and found no impact of packet sizes that are not multiples of the CPU's word or cache line size. The throughput scales sub-linearly with the number of flows as the NIC distributes the flows to different CPU cores. Adding an additional flow increases the performance by about 90% until all four cores of the CPU are utilized.

As we observed linear scaling with earlier versions of OvS we investigated further. Figure 5 compares the throughput and scaling with flows of all recent versions of OvS that are compatible with Linux kernel 3.7. Versions prior to 1.11.0 scale linearly whereas later versions only scale sub-linearly, i.e. adding an additional core does not increase the throughput by 100% of the single flow throughput. Profiling reveals that this is due to a contended spin lock that is used to synchronize access to statistics counters for the flows. Later versions support wild card flows in the kernel and match the whole synthetic test traffic to a single wildcarded datapath rule in this scenario. So all packets of the different flows use the same statistics counters, this leads to a lock contention. A realistic scenario with multiple rules or more (virtual) network ports does not exhibit this behavior. Linear scaling with the number of CPU cores can, therefore, be assumed in real-world scenarios and further tests are restricted to a single CPU core. The throughput per core is 1.88 Mpps.

5.3 Larger Number of Flows

We derive a test case from the OvS architecture described in Section 2.3: Testing more than four flows exercises the flow table lookup and update mechanism in the kernel module due to increased flow table size. The generated flows for this test use different layer 2 addresses to avoid the generation of wild card rules in the OvS datapath kernel module. This simulates a switch with multiple attached devices.

Figure 6 shows that the total throughput is affected by the number of flows due to increased cache misses during the flow table lookup. The total throughput drops from about 1.87 Mpps[2] with a single flow to 1.76 Mpps with 2000 flows. The interrupts were restricted to a single CPU core.

Another relevant scenario for a cloud system is cloning a flow and sending it to multiple output destinations, e.g., to forward traffic to an intrusion detection system or to implement multicast. Figure 7 shows that performance drops by 30% when a flow is sent out twice and another 25% when it is copied one more time. This demonstrates that a large amount of the performance can be attributed to packet I/O and not processing. About 30% of the CPU time is spent in the driver and network stack sending packets. This needs to be considered when a monitoring system is to be integrated into a system involving software switches. An intrusion detection system often works via passive monitoring of mirrored traffic. Hardware switches can do this without overhead in hardware, but this is a significant cost for a software switch.

5.4 Open vSwitch Throughput with Virtual Network Interfaces

Virtual network interfaces exhibit different performance characteristics than physical interfaces. For example, dropping packets in an overload condition is done efficiently and concurrently in hardware on a pNIC whereas a vNIC needs to drop packets in software. We, therefore, compare the performance of the pNIC to pNIC forwarding with the pNIC to vNIC scenario shown in Figure 3a.

Figure 8 compares the observed throughput under increasing offered load with both physical and virtual interfaces. The graph for traffic sent into a VM shows an inflection point at an offered load of 0.5 Mpps. The throughput then continues to increase until it reaches 0.85 Mpps, but a constant ratio of the incoming packets is dropped. This start of drops is accompanied by a sudden increase in CPU load in the kernel. Profiling the kernel with `perf` shows that this is caused by increased context switching and functions related to packet queues. Figure 9 plots the CPU load caused by context switches (kernel function `__switch_to`) and functions related to virtual NIC queues at the tested offered loads with a run time of five minutes per run. This indicates that a congestion occurs at the vNICs and the system tries to resolve this by forcing a context switch to the network task of the virtual machine to retrieve the packets. This additional overhead leads to drops.

[2] Lower than the previously stated figure of 1.88 Mpps due to active profiling.

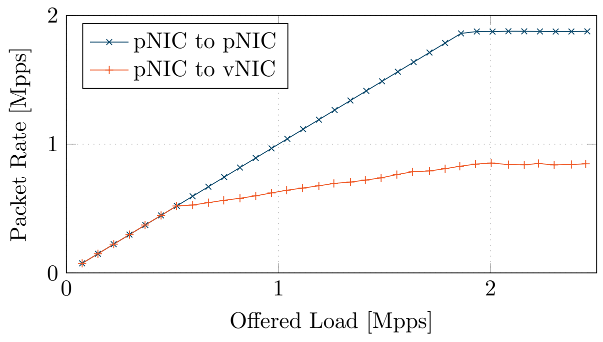


Fig. 8 Offered load vs. throughput with pNICs and vNICs

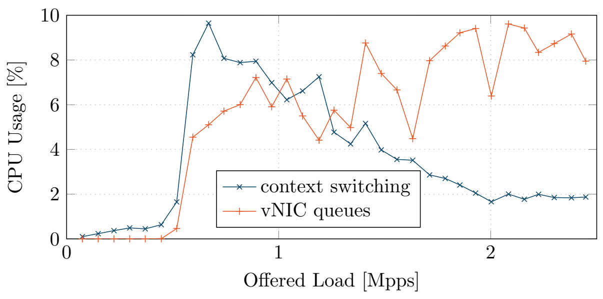


Fig. 9 CPU load of context switching and vNIC queuing

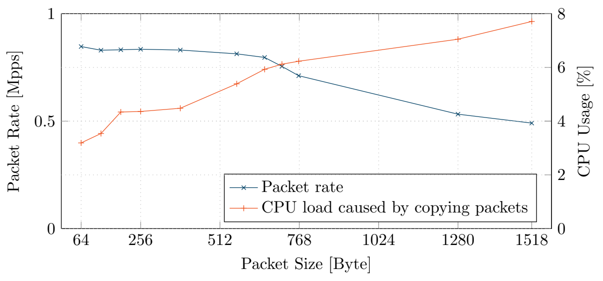


Fig. 10 Packet size vs. throughput and memory copy overhead with vNICs

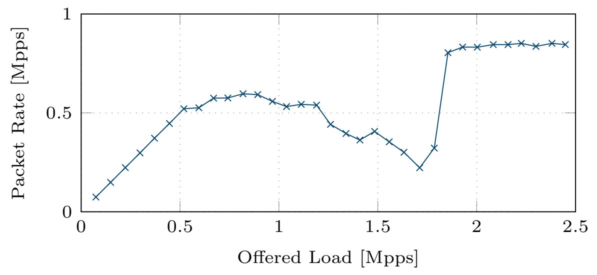


Fig. 11 Throughput without explicitly pinning all tasks to CPU cores

Packet sizes are also relevant in comparison to the pNIC to pNIC scenario because the packet needs to be copied to the user space to forward it to a VM. Figure 10 plots the throughput and the CPU load of the kernel function `copy_user_enhanced_fast_string`, which copies a packet into the user space, in the forwarding scenario shown in Figure 3a. The throughput

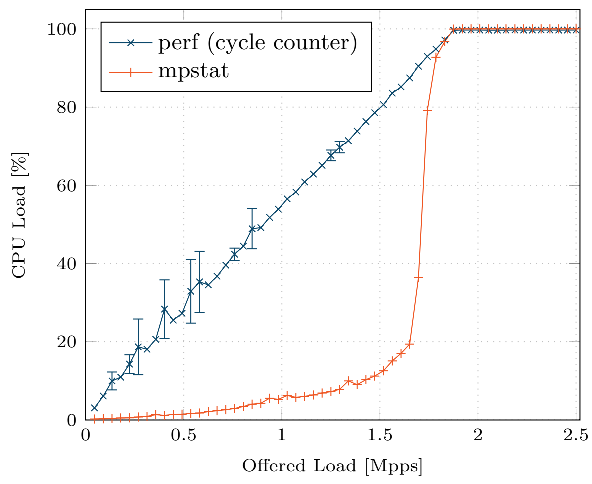


Fig. 12 CPU load when forwarding packets with Open vSwitch

drops only marginally from 0.85 Mpps to 0.8 Mpps until it becomes limited by the line rate with packets larger than 656 Byte. Copying packets poses a measurable but small overhead. The reason for this is the high memory bandwidth of modern servers: our test server has a memory bandwidth of 200 Gbit per second. This means that VMs are well-suited for running network services that rely on bulk throughput with large packets, e.g., file servers. Virtualizing packet processing or forwarding systems that need to be able to process a large number of small packets per second is, however, problematic.

We derive another test case from the fact that the DuT runs multiple applications: OvS and the VM receiving the packets. This is relevant on a virtualization server where the running VMs generate substantial CPU load. The VM was pinned to a different core than the NIC interrupt for the previous test. Figure 11 shows the throughput in the same scenario under increasing offered load, but without pinning the VM to a core. This behavior can be attributed to a scheduling conflict because the Linux kernel does not measure the load caused by interrupts properly by default. Figure 12 shows the average CPU load of a core running only OvS as seen by the scheduler (read from the `procfs` pseudo filesystem with the `mpstat` utility) and compares it to the actual average load measured by reading the CPU's cycle counter with the profiling utility `perf`.

Guideline 2 Virtual machine cores and NIC interrupts should be pinned to disjoint sets of CPU cores.

The Linux scheduler does not measure the CPU load caused by hardware interrupts properly and therefore schedules the VM on the same core, which impacts the performance. `CONFIG_IRQ_TIME_ACCOUNTING` is a kernel option, which can be used to enable accurate reporting of CPU usage by interrupts, which resolves this conflict. However, this option is not enabled by default in the Linux kernel because it slows down interrupt handlers, which are designed to be executed as fast as possible.

Guideline 3 CPU load of cores handling interrupts should be measured with hardware counters using `perf`.

We conducted further tests in which we sent external traffic through a VM and into a different VM or to another pNIC as shown in Figure 3b and 3c in Section 4. The graphs for the results of more detailed tests in these scenarios provide no further insight beyond the already discussed results from this section because sending and receiving traffic from and to a vNIC show the same performance characteristics.

5.5 Conclusion

Virtual switching is limited by the number of packets, not the overall throughput. Applications that require a large number of small packets, e.g., virtualized network functions, are thus more difficult for a virtual switch than applications relying on bulk data transfer, e.g., file servers. Overloading virtual ports on the switch can lead to packet loss before the maximum throughput is achieved.

Using the DPDK backend in OvS can improve the throughput by a factor of 7 when no VMs are involved. With VMs, an improvement of a factor of 3 to 4 can be achieved, cf. Table 1. However, DPDK requires statically assigned CPU cores that are constantly being utilized by a busy-wait polling logic, causing 100% load on these cores. Using the slower default Linux IO backend results in a linear correlation between network load and CPU load, cf. Figure 12.

6 Latency Measurements

In another set of measurements we address the packet delay introduced by software switching in OvS. Therefore, we investigate two different scenarios. In the first experiment, traffic is forwarded between two physical interfaces (cf. Figure 3d). For the second scenario the packets are not forwarded between the physical interfaces directly but through a VM as it is shown in Figure 3b.

6.1 Forwarding between Physical Interfaces

Figure 13 shows the measurement for a forwarding between two pNICs by Open vSwitch. This graph features

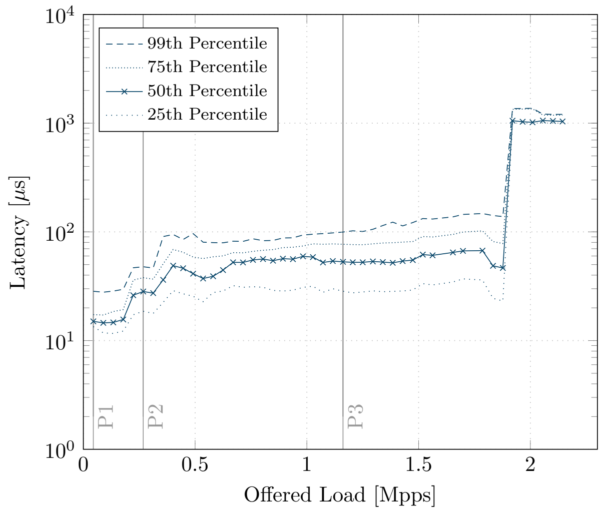


Fig. 13 Latency of packet forwarding between pNICs

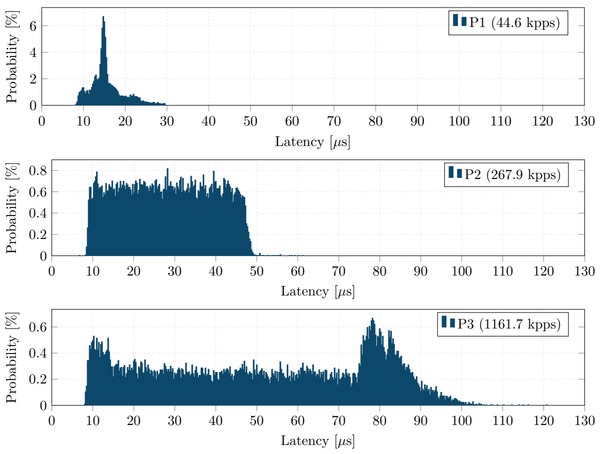


Fig. 14 Latency distribution for forwarding between pNICs

four different levels of delay. The first level has an average latency of around 15 μs and a packet transfer rate of up to 179 kpps. Above that transfer rate the second level has a delay value of around 28 μs and lasts up to 313 kpps. Beyond that rate the third level offers a latency of around 53 μs up to a transfer rate of 1.78 Mpps. We selected three different points (P1 - P3) – one as a representative for each of the levels before the system becomes overloaded (cf. Figure 13). Table 2 also includes these points to give typical values for their corresponding level.

The reason for the shape and length of the first three levels is the architecture of the `ixgbe` driver as described by Beifuß et al. [11]. This driver limits the interrupt rate to 100k per second for packet rates lower than 156.2 kpps, which relates the highest transfer rate measured for the first level in Figure 13. The same observation holds for the second and the third level. The interrupt rate is limited to 20k per second for transfer rates lower than 312.5 kpps, and to 8k per second above that. These packet rates equal to the step into the next plateau of the graph.

At the end of the third level the latency drops again right before the switch is overloaded. Note that the drop in latency occurs at the point at which the Linux scheduler begins to recognize the CPU load caused by interrupts (cf. Section 5.4, Figure 12). The Linux scheduler is now aware that the CPU core is almost fully loaded with interrupt handlers and therefore stops scheduling other tasks on it. This causes a slight decrease in latency. Then the fourth level is reached and the latency increases to about 1 ms as Open vSwitch can no longer cope with the load and all queues fill up completely.

We visualized the distributions of latency at three measurement points *P1* – *P3* (cf. Figure 13 and Table 2). The distributions at these three measurements are plotted as histogram with bin width of 0.25 μs in Figure 14. The three selected points show the typical shapes of the probability density function of their respective levels.

At *P1* the distribution shows the behavior before the interrupt throttle rate affects processing, i.e. one interrupt per packet is used. The latencies are approximately normally distributed as each packet is processed independently.

The distribution at *P2* demonstrates the effect of the ITR used by the driver. A batch of packets accumulates on the NIC and is then processed by a single interrupt. This causes a uniform distribution as each packet is in a random position in the batch.

For measurement *P3* the distribution depicts a high load at which both the interrupt throttle rate and the poll mechanism of the NAPI affect the distribution. A significant number of packets accumulates on the NIC before the processing is finished. Linux then polls the NIC again after processing, without re-enabling interrupts in between, and processes a second smaller batch. This causes an overlay of the previously seen uniform distribution with additional peaks caused by the NAPI processing.

Overloading the system leads to an unrealistic excessive latency of ≈ 1 ms and its exact distribution is of little interest. Even the best-case 1st percentile shows a latency of about 375 μs in all measurements during overload conditions, far higher than even the worst-case of the other scenarios.

Guideline 4 Avoid overloading ports handling latency-critical traffic.

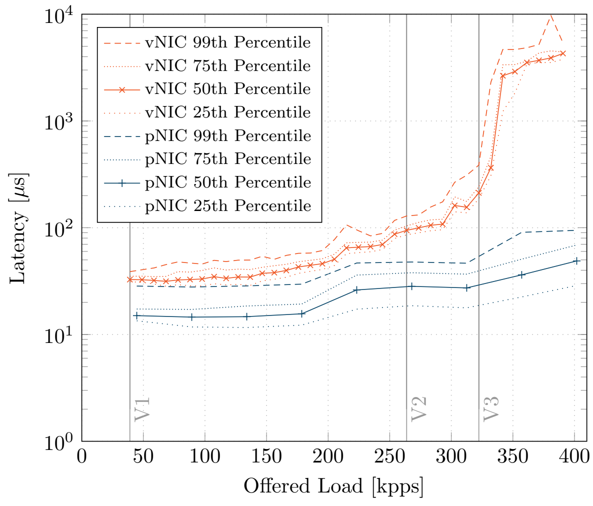


Fig. 15 Latency of packet forwarding through VM

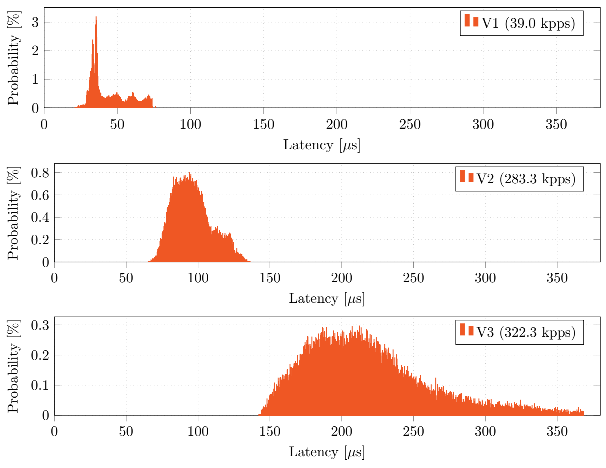


Fig. 16 Latency distribution of packet forwarding through VM

6.2 Forwarding Through Virtual Machines

For delay measurements of VMs we use our setup as depicted in Figure 3b. There the traffic originating from the measurement server is forwarded through the VM and back to the measurement server.

Figure 15 compares the latency in this scenario with the pNIC forwarding from Section 6.1. Note that the plot uses a logarithmic y-axis, so the gap between the slowest and the fastest packets for the vNIC scenario is wider than in the pNIC scenario even though it appears smaller.

The graph for the vNICs does not show plateaus of steady latency like the pNIC graph but a rather smooth growth of latency. Analogous to P1 – P3 in the pNIC scenario, we selected three points V1 – V3 as depicted in Figure 15. Each point is representative to the relating levels of latency. The three points are also available in Table 2. The development of the latency under increasing load shows the same basic characteristics as in the pNIC scenario due to the interrupt-based processing of the incoming packets. However, the additional work-load and the emulated NICs smooth the sharp inflection points and also increase the delay.

The histograms for the latency at the lowest investigated representatively selected packet rate – P1 in Figure 14 and V1 in Figure 16 – have a similar shape.

The shape of the histogram in V3 is a long-tail distribution, i.e. while the average latency is low, there is a significant number of packets with a high delay. This distribution was also observed by Whiteaker et al. [54] in virtualized environments. However, we could only observe this type of traffic under an overload scenario like V3. Note that the maximum packet rate for this scenario was previously given as 300 kpps in Table 1, so V3 is already an overload scenario. We could not observe such a distribution under normal load. The worst-case latency is also significantly higher than in the pNIC scenario due to the additional buffers in the vNICs.

Guideline 5 Avoid virtualizing services that are sensitive to a high 99th percentile latency.

6.3 Improving Latency with DPDK

Porting OvS to DPDK also improves the latency. Figure 17 visualizes representative histograms of latency probability distributions of forwarding between physical and virtual interfaces.

DPDK uses a busy-wait loop polling all NICs instead of relying on interrupts, resulting in the previously mentioned constant CPU load of 100% on all assigned cores. The resulting latency is thus a normal distribution and there are no sudden changes under increasing load as no interrupt moderation algorithms are used. We measured a linearly increasing median latency from 7.8 μs (99th percentile: 8.2 μs) at 0.3 Mpps to 13.9 μs (99th percentile: 23.0 μs) at 13.5 Mpps when forwarding between two physical network interfaces.

Adding virtual machines leads again to a long-tail distribution as the traffic is processed by multiple different processes on different queues. The virtual machine still uses the VirtIO driver which also does not feature sophisticated interrupt adaptation algorithms. Hence, the distribution stays stable regardless of the applied load. The overall median latency stayed with the range of 13.5 μs to 14.1 μs between 0.03 Mpps and 1 Mpps. However, the 99th percentile increases from 15 μs Mpps to 150 μs over the same range, i.e. the long tail grows longer as the load increases.

Table 2 Comparison of Latency

Scenario	Load [kpps]	Load* [%]	Average [μs]	Std.Dev. [μs]	25th Perc. [μs]	50th Perc. [μs]	95th Perc. [μs]	99th Perc. [μs]
P1 (pNIC)	44.6	2.3	15.8	4.6	13.4	15.0	17.3	23.8
P2 (pNIC)	267.9	14.0	28.3	11.2	18.6	28.3	37.9	45.8
P3 (pNIC)	1161.7	60.5	52.2	27.0	28.5	53.0	77.0	89.6
V1 (vNIC)	39.0	11.4	33.0	3.3	31.2	32.7	35.1	37.3
V2 (vNIC)	283.3	82.9	106.7	16.6	93.8	105.5	118.5	130.8
V3 (vNIC)	322.3	94.3	221.1	49.1	186.7	212.0	241.9	319.8

*) Normalized to the load at which more than 10% of the packets were dropped, i.e. a load $\geq$ 100% would indicate an overload scenario

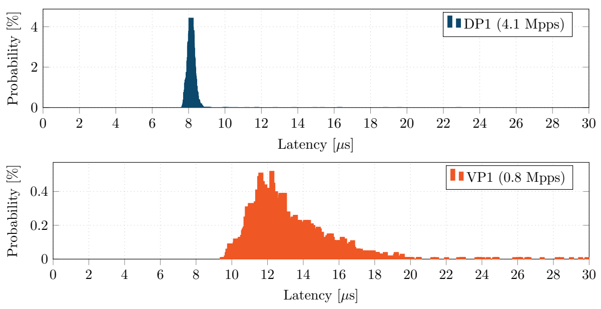


Fig. 17 Latency distribution of forwarding with DPDK

6.4 Conclusion

Latency depends on the load of the switch. This effect is particularly large when OvS is running in the Linux kernel due to interrupt moderation techniques leading to changing latency distributions as the load increases. Overloading a port leads to excessive worst-case latencies that are one to two orders of magnitude worse than latencies before packet drops occur. Virtual machines exhibit a long-tail distribution of the observed latencies under high load. These problems can be addressed by running OvS with DPDK which exhibits a more consistent and lower latency profile.

7 Conclusion

We analyzed the performance characteristics and limitations of the Open vSwitch data plane, a key element in many cloud environments. Our study showed good performance when compared to other Linux kernel forwarding techniques, cf. Section 5.1. A few guidelines for system operators can be derived from our results:

Guideline 1 To improve performance, OvS should be preferred over the default Linux tools when using cloud frameworks. Consider DPDK als backend for OvS for future deployments.

Guideline 2 Virtual machine cores and NIC interrupts should be pinned to disjoint sets of CPU cores. Figure 11 shows performance drops when no pinning is used. The load caused by processing packets on the hypervisor should also be considered when allocating CPU resources to VMs. Even a VM with only one virtual CPU core can load two CPU cores due to virtual switching. The total system load of Open vSwitch can be limited by restricting the NIC's interrupts to a set of CPU cores instead of allowing them on all cores. If pinning all tasks is not feasible, make sure to measure the CPU load caused by interrupts properly.

Guideline 3 CPU load of cores handling interrupts should be measured with hardware counters using `perf`. The kernel option `CONFIG_IRQ_TIME_ACCOUNTING` can be enabled despite its impact on the performance of interrupt handlers, to ensure accurate reporting of CPU utilization with standard tools, cf. Figure 12. Note that the performance of OvS is not impacted by this option as the Linux kernel prefers polling over interrupts under high load.

Guideline 4 Avoid overloading ports handling latency-critical traffic. Overloading a port impacts latency by up to two orders of magnitude due to buffering in software. Hence, bulk traffic should be kept separated from latency-critical traffic.

Guideline 5 Avoid virtualizing services that are sensitive to a high 99th percentile latency. Latency doubles when using a virtualized application compared to a natively deployed application, cf. Section 6. This is usually not a problem as the main share of latency is caused by the network and not by the target server. However, the worst-case latency (99th percentile) for packets increases by an order of magnitude for packets processed by a VM, cf. Section 6.2. This can be problematic for protocols with real-time requirements.

Virtualizing services that rely on bulk data transfer via large packets, e.g., file servers, achieve a high throughput measured in Gbit/s, cf. Figure 10. The per-packet overhead dominates over the per-byte overhead. Services relying on smaller packets are thus more difficult to handle. Not only the packet throughput suffers from virtualization: latency also increases by a factor of 2 and the 99th percentile even by an order of magnitude. However, moving packet processing systems or virtual switches and routers into VMs is problematic because of the high overhead per packet that needs to cross the VM/host barrier and because of their latency-sensitive nature.

The shift to user space packet-processing frameworks like DPDK promises substantial improvements for both throughput (cf. Section 5.1) and latency (cf. Section 6). DPDK is integrated, but disabled by default, in Open vSwitch. However, the current version we evaluated still had stability issues and is not yet fit for production. Further issues with the DPDK port are usability as complex configuration is required and the lack of debugging facilities as standard tools like tcpdump are currently not supported. Intel is currently working on improving these points to get DPDK vSwitch into production [30].

Acknowledgments

This research has been supported by the DFG (German Research Foundation) as part of the MEMPHIS project (CA 595/5-2) and in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1), partly funded by the German BMBF (Project ID 16KIS0460K). The authors alone are responsible for the content of the paper.

References

1. Intel DPDK: Data Plane Development Kit. `http://dpdk.org`. Last visited 2016-03-27
2. Intel DPDK vSwitch. `https://01.org/sites/default/files/page/intel_dpdk_vswitch_performance_figures_0.10.0_0.pdf`. Last visited 2016-03-27
3. Intel DPDK vSwitch. `https://github.com/01org/dpdk-ovs`. Last visited 2016-03-27
4. Intel I/O Acceleration Technology. `http://www.intel.com/content/www/us/en/wireless-network/accel-technology.html`. Last visited 2016-03-27
5. Open vSwitch. `http://openvswitch.org`. Last visited 2016-03-27
6. OpenNebula. `https://opennebula.org`. Last visited 2016-03-27
7. OpenStack. `https://openstack.org`. Last visited 2016-03-27
8. Virtual Machine Device Queues: Technical White Paper (2008)
9. Impressive Packet Processing Performance Enables Greater Workload Consolidation (2013)
10. Angrisani, L., Ventre, G., Peluso, L., Tedesco, A.: Measurement of Processing and Queuing Delays Introduced by an Open-Source Router in a Single-Hop Network. IEEE Transactions on Instrumentation and Measurement **55**(4), 1065–1076 (2006)
11. Beifuß, A., Raumer, D., Emmerich, P., Runge, T.M., Wohlfart, F., Wolfinger, B.E., Carle, G.: A Study of Networking Software Induced Latency. In: 2nd International Conference on Networked Systems 2015. Cottbus, Germany (2015)
12. Bianco, A., Birke, R., Giraudo, L., Palacin, M.: Openflow switching: Data plane performance. In: International Conference on Communications (ICC). IEEE (2010)
13. Bolla, R., Bruschi, R.: Linux Software Router: Data Plane Optimization and Performance Evaluation. Journal of Networks **2**(3), 6–17 (2007)
14. Bradner, S., McQuaid, J.: Benchmarking Methodology for Network Interconnect Devices. RFC 2544 (Informational) (1999)
15. Cardigliano, A., Deri, L., Gasparakis, J., Fusco, F.: vPFRING: Towards WireSpeed Network Monitoring using Virtual Machines. In: ACM Internet Measurement Conference (2011)
16. Deri, L.: nCap: Wire-speed Packet Capture and Transmission. In: IEEE Workshop on End-to-End Monitoring Techniques and Services, pp. 47–55 (2005)
17. Dobrescu, M., Argyraki, K., Ratnasamy, S.: Toward Predictable Performance in Software Packet-Processing Platforms. In: USENIX Conference on Networked Systems Design and Implementation (NSDI) (2012)
18. Dobrescu, M., Egi, N., Argyraki, K., Chun, B., Fall, K., Iannaccone, G., Knies, A., Manesh, M., Ratnasamy, S.: RouteBricks: Exploiting Parallelism To Scale Software Routers. In: 22nd ACM Symposium on Operating Systems Principles (SOSP) (2009)
19. DPDK Project: DPDK 16.11 Release Notes. `http://dpdk.org/doc/guides/rel_notes/release_16_11.html` (2016). Last visited 2016-03-27
20. Emmerich, P., Gallenmüller, S., Raumer, D., Wohlfart, F., Carle, G.: MoonGen: A Scriptable High-Speed Packet Generator. In: 15th ACM SIGCOMM Conference on Internet Measurement (IMC'15) (2015)
21. Emmerich, P., Raumer, D., Wohlfart, F., Carle, G.: A Study of Network Stack Latency for Game Servers. In: 13th Annual Workshop on Network and Systems Support for Games (NetGames'14). Nagoya, Japan (2014)

22. Emmerich, P., Raumer, D., Wohlfart, F., Carle, G.: Performance Characteristics of Virtual Switching. In: 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet'14). Luxembourg (2014)
23. ETSI: Network Functions Virtualisation (NFV); Architectural Framework, V1.1.1 (2013)
24. Han, S., Jang, K., Panda, A., Palkar, S., Han, D., Ratnasamy, S.: Softnic: A software nic to augment hardware. Tech. Rep. UCB/EECS-2015-155, EECS Department, University of California, Berkeley (2015)
25. He, Z., Liang, G.: Research and evaluation of network virtualization in cloud computing environment. In: Networking and Distributed Computing (ICNDC), pp. 40–44. IEEE (2012)
26. Huggahalli, R., Iyer, R., Tetrick, S.: Direct Cache Access for High Bandwidth Network I/O. In: Proceedings of the 32nd Annual International Symposium on Computer Architecture, pp. 50–59 (2005)
27. Hwang, J., Ramakrishnan, K.K., Wood, T.: Netvm: High performance and flexible networking using virtualization on commodity platforms. In: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp. 445–458. USENIX Association, Seattle, WA (2014)
28. Jarschel, M., Oechsner, S., Schlosser, D., Pries, R., Goll, S., Tran-Gia, P.: Modeling and performance evaluation of an OpenFlow architecture. In: Proceedings of the 23rd International Teletraffic Congress. ITCP (2011)
29. Kang, N., Liu, Z., Rexford, J., Walker, D.: Optimizing the "One Big Switch" Abstraction in Software-defined Networks. In: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT '13, pp. 13–24. ACM, New York, NY, USA (2013). DOI 10.1145/2535372.2535373. URL `http://doi.acm.org/10.1145/2535372.2535373`
30. Kevin Traynor: OVS, DPDK and Software Dataplane Acceleration. `https://fosdem.org/2016/schedule/event/ovs_dpdk/attachments/slides/1104/export/events/attachments/ovs_dpdk/slides/1104/ovs_dpdk_fosdem_16.pdf` (2016). Last visited 2016-03-27
31. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F.: The Click Modular Router. ACM Transactions on Computer Systems (TOCS) **18**(3), 263–297 (2000). DOI 10.1145/354871.354874
32. Larsen, S., Sarangam, P., Huggahalli, R., Kulkarni, S.: Architectural Breakdown of End-to-End Latency in a TCP/IP Network. International Journal of Parallel Programming **37**(6), 556–571 (2009)
33. Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., Huici, F.: ClickOS and the Art of Network Function Virtualization. In: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp. 459–473. USENIX Association, Seattle, WA (2014)
34. Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., Huici, F.: Clickos and the art of network function virtualization. In: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp. 459–473. USENIX Association, Seattle, WA (2014)
35. Meyer, T., Wohlfart, F., Raumer, D., Wolfinger, B., Carle, G.: Validated Model-Based Prediction of Multi-Core Software Router Performance. Praxis der Informationsverarbeitung und Kommunikation (PIK) (2014)
36. Michael Tsirkin Cornelia Huck, P.M.: Virtual I/O Device (VIRTIO) Version 1.0 Committee Specification 04. OASIS (2016)
37. Munch, B.: Hype Cycle for Networking and Communications. Report, Gartner (2013)
38. Niu, Z., Xu, H., Tian, Y., Liu, L., Wang, P., Li, Z.: Benchmarking NFV Software Dataplanes **arXiv:1605.05843** (2016)
39. OpenStack: Networking Guide: Deployment Scenarios. `http://docs.openstack.org/liberty/networking-guide/deploy.html` (2015). Last visited 2016-03-27
40. Panda, A., Han, S., Jang, K., Walls, M., Ratnasamy, S., Shenker, S.: Netbricks: Taking the v out of nfv. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 203–216. USENIX Association, GA (2016)
41. Pettit, J., Gross, J., Pfaff, B., Casado, M., Crosby, S.: Virtual Switching in an Era of Advanced Edges. In: 2nd Workshop on Data Center Converged and Virtual Ethernet Switching (DC-CAVES) (2011)
42. Pfaff, B., Pettit, J., Koponen, T., Amidon, K., Casado, M., Shenker, S.: Extending Networking into the Virtualization Layer. In: Proc. of workshop on Hot Topics in Networks (HotNets-VIII) (2009)
43. Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., Casado, M.: The design and implementation of open vswitch. In: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). USENIX Association (2015)
44. Pongracz, G., Molnar, L., Kis, Z.L.: Removing Roadblocks from SDN: OpenFlow Software Switch Performance on Intel DPDK. Second European Workshop on Software Defined Networks

(EWSDN'13) pp. 62–67 (2013)
45. Ram, K.K., Cox, A.L., Chadha, M., Rixner, S.: Hyper-Switch: A Scalable Software Virtual Switching Architecture. In: Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13), pp. 13–24. USENIX, San Jose, CA (2013)
46. Rizzo, L.: netmap: a novel framework for fast packet I/O. In: USENIX Annual Technical Conference (2012)
47. Rizzo, L., Carbone, M., Catalli, G.: Transparent Acceleration of Software Packet Forwarding using Netmap. In: INFOCOM, pp. 2471–2479. IEEE (2012)
48. Rizzo, L., Lettieri, G.: VALE, a switched ethernet for virtual machines. In: C. Barakat, R. Teixeira, K.K. Ramakrishnan, P. Thiran (eds.) CoNEXT, pp. 61–72. ACM (2012)
49. Rotsos, C., Sarrar, N., Uhlig, S., Sherwood, R., Moore, A.W.: Oflops: An Open Framework for OpenFlow Switch Evaluation. In: Passive and Active Measurement, pp. 85–95. Springer (2012)
50. Salim, J.H., Olsson, R., Kuznetsov, A.: Beyond Softnet. In: Proceedings of the 5th annual Linux Showcase & Conference, vol. 5, pp. 18–18 (2001)
51. Tedesco, A., Ventre, G., Angrisani, L., Peluso, L.: Measurement of Processing and Queuing Delays Introduced by a Software Router in a Single-Hop Network. In: IEEE Instrumentation and Measurement Technology Conference, pp. 1797–1802 (2005)
52. Thomas Monjalon: dropping librte_ivshmem. `http://dpdk.org/ml/archives/dev/2016-June/040844.html` (2016). Mailing list discussion
53. Wang, G., Ng, T.E.: The impact of virtualization on network performance of amazon ec2 data center. In: INFOCOM, pp. 1–9. IEEE (2010)
54. Whiteaker, J., Schneider, F., Teixeira, R.: Explaining Packet Delays under Virtualization. ACM SIGCOMM Computer Communication Review **41**(1), 38–44 (2011)

Author Biographies

Paul Emmerich is a Ph.D. student at the Chair for Network Architectures and Services at Technical University of Munich. He received his M.Sc. in Informatics at the Technical University of Munich in 2014. His research interests include packet generation as well as software switches and routers.

Daniel Raumer is a Ph.D. student at the Chair of Network Architectures and Services at Technical University of Munich, where he received his B.Sc. and M.Sc. in Informatics, in 2010 and 2012. He is concerned with device performance measurements with relevance to Network Function Virtualization as part of Software-defined Networking architectures.

Sebastian Gallenmüller is a Ph.D. student at the Chair of Network Architectures and Services at Technical University of Munich. There he received his M.Sc. in Informatics in 2014. He focuses on the topic of assessment and evaluation of software systems for packet processing.

Florian Wohlfart is a Ph.D. student working at the Chair of Network Architectures and Services at Technical University of Munich. He received his M.Sc. in Informatics at Technical University of Munich in 2012. His research interests include software packet processing, middlebox analysis, and network performance measurements.

Georg Carle is professor at the Department of Informatics at Technical University of Munich, holding the Chair of Network Architectures and Services. He studied at University of Stuttgart, Brunel University, London, and Ecole Nationale Superieure des Telecommunications, Paris. He did his Ph.D. in Computer Science at University of Karlsruhe, and worked as postdoctoral scientist at Institut Eurecom, Sophia Antipolis, France, at the Fraunhofer Institute for Open Communication Systems, Berlin, and as professor at the University of Tübingen.