Accepted at 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)

Performance Characteristics of Virtual Switching

Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle

Technische Universität München, Department of Computer Science, Network Architectures and Services
{emmericp|raumer|wohlfart|carle}@net.in.tum.de

***Abstract*—Virtual switches, like Open vSwitch, have emerged as an important part of cloud networking architectures. They connect interfaces of virtual machines and establish the connection to the outer network via physical network interface cards. Today, all important cloud frameworks support Open vSwitch as the default virtual switch. However, general understanding about the performance implications of Open vSwitch in different usage scenarios is missing. In this work we provide insights into the performance properties by systematically conducting measurements in virtual switching setups. We present quantitative and qualitative performance results of Open vSwitch in scenarios involving physical and virtual network interfaces.**

I. 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 lead to the term *virtual switches* and paved the way for the mainstream adoption of software switches [1], 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 limitations. The most popular virtual switch implementation – *Open vSwitch* (OvS) – is heavily used in cloud computing frameworks like OpenStack [2] and OpenNebula [3]. OvS is an open source project that is backed by an active community, and supports common standards such as OpenFlow, SNMP, and IPFIX.

Efficiency of packet processing in software has multiple dependencies, and each factor can significantly hurt the performance, which gives 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.

The remainder of this paper is structured as follows: Section II provides an overview on software switching. We explain recent developments in hardware and software that enables sufficient performance in general purpose PC systems based on commodity hardware, highlight challenges, and provide an overview of Open vSwitch. We then present related work on performance measurements in Section III. Section IV describes our study on the performance of software switches. Section V sums up our results and gives advice for the deployment of software switches.

II. Software Switches

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 referred to as virtual switches. Compared to the default VM bridging solutions, virtual switches like OvS are more flexible and provide a whole range of additional features.

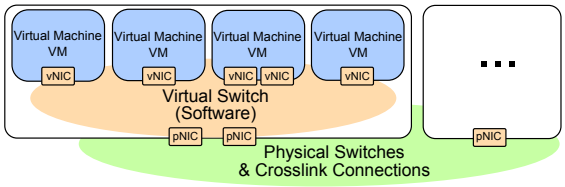


Fig. 1. Application scenario of a virtual switch

Figure 1 illustrates the typical application scenario of virtual switches. The switch connects the virtual network interface cards (NIC) `vNIC` and the physical NICs `pNIC`. Typical applications in virtual host environments include traffic switching from pNIC to vNIC, vNIC to pNIC, and vNIC to vNIC. As components of future network architectures packet flows traversing a chain of VMs are also discussed [4]. 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 still the dominant way to inter-connect physical machines, software switches 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. The size of flow tables in software switches is just a matter of their configuration while it is limited to a few thousand in hardware switches [29].

A. 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 mecha-

nisms are built into the network adapters: checksum calculations and distribution of packets directly to the addressed VM [5]. NICs can transfer packets into memory (DMA) and even into the CPU caches (DCA) [6] without involving the CPU. These techniques can eliminate workloads of hundreds of CPU cycles per packet. 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 in-kernel 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 software 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 [7], [8], [9], [10] features like RSS and offloading are important to reduce CPU load and help to distribute load among the available cores.

Packet forwarding applications like Open vSwitch [11], the Linux router, or Click Modular Router [12] 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 like 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 [13], Netmap [14], and DPDK [15], [16]. E.g. with DPDK the L3 forwarding performance on an Intel Xeon E5645 (6x 2.4 GHz cores) achieves 35.2 Mpps [15]. Virtual switches like VALE [17] achieve over 17 Mpps vNIC to vNIC bridging performance by utilizing shared memory between VMs and the hypervisor. Similar prototypes to VALE exist [18], [4]. Virtual switches in combination with guest OSes like ClickOS [4] achieve notable performance of packet processing in VMs. All these techniques rely on modified drivers, VM environments, and network stacks, and only work with certain NICs. Experiments which combine OvS with the described high speed packet processing frameworks [19], [20] demonstrate performance improvements. In our experiments we encountered a high instability of these high speed packet processing frameworks when VMs are involved. We do not consider these frameworks production-ready for virtual switches. We therefore focus our measurements on virtual switches which are used in real-world scenarios – e.g. in cloud frameworks.

B. Open vSwitch

Open vSwitch [11], [21], [22] is an OpenFlow switch implementation that can be used both as a pure virtual switch in virtualized environments and as a general purpose software switch that connect physically separated nodes.

Figure 2 illustrates the different processing paths in OvS. The two most important components are the switch daemon `ovs-vswitchd` that controls the switch and implements the OpenFlow protocol, and the *datapath*, a kernel module that

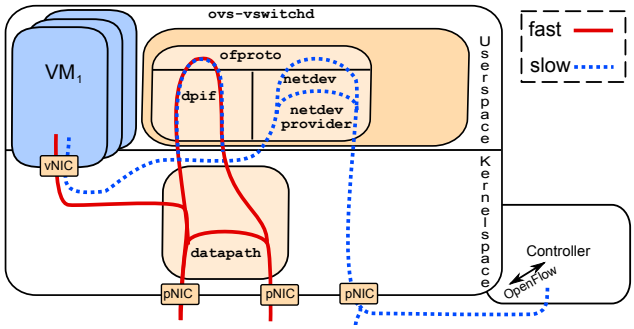


Fig. 2. Open vSwitch architecture representing the data processing flows

implements the actual packet forwarding. The datapath kernel module processes packets with 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 or dropping all packets from or to a specific IP address. These rules are called flows but they are different from OpenFlow rules as one design choice of the kernel module was to keep it as simple as possible in order to achieve a high performance [22].

A packet that can be processed by a rule in the datapath 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 red line. Packets that do not match any flow in the flow table are forced on the *slow path* (dotted line), which copies the packet to the user space and forwards it to the OvS daemon. 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 `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 like a cache for the bigger and more complicated OpenFlow flow table in the slow path.

III. Related Work

Detailed performance analysis of PC-based packet processing systems have been continuously addressed in the past. In 2007, Bolla and Bruschi [9] applied both external and internal measurements for an analysis of a Linux 2.6[1] software router. The RouteBricks project [7] revealed performance influences of multi-core PC systems and of different workloads [23].

In the context of different modifications to the guest and host OS network stack (cf. Section II-A), virtual switching performance was measured [17], [4], [24], [19], [20] but provides only limited possibility for direct comparison. Other

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

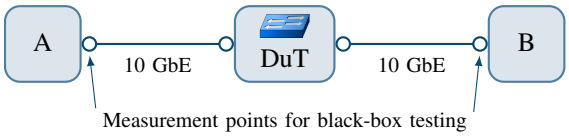


Fig. 3. Server setup

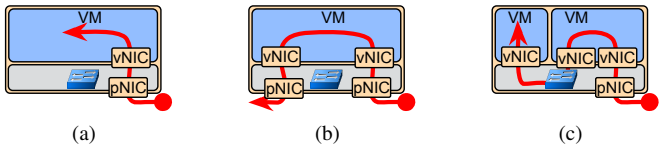


Fig. 4. Test setups with virtual machines

studies addressed the performance of virtual switching within a performance analysis of cloud datacenters [25], but provide less detailed information on virtual switching performance.

Both papers of the OvS developers [21], [22] only provide coarse measurements of throughput performance in bits per second in vNIC to vNIC switching scenarios. Neither frame lengths nor measurement results in pps are provided. In [26] 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 [27]. 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. In [29] OFLOPS, a framework for OpenFlow switch evaluation is presented and amongst others applied to OvS. 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.

IV. Software Switching Performance

We ran tests to quantify the performance of several software switches with a focus on OvS in scenarios involving both physical and virtual network interfaces.

A. Test Methodology

Our test setup is shown in Fig. 3. Servers A and B are used as load generators and packet counters, the DuT (Device under Test) runs the software under test. In black-box tests we avoid any overhead on the DuT through measurements, so we measure the offered load and the throughput on A and B. The DuT runs the Linux tool `perf` for white-box tests; this overhead reduces the maximum throughput by $\sim 1\%$.

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 DPDK vSwitch 0.10 with manually created OpenFlow rules, and qemu-kvm 1.1.2 with VirtIO network adapters unless mentioned otherwise. Our load generator uses the PF_RING DNA [13] framework and is able to generate minimally sized UDP packets at line rate on 10 GBit interfaces (14.88 Mpps). The sink periodically samples the statistics registers of the NIC to measure the throughput.

The DuT uses an Intel X520-SR2 dual 10 GbE network interface card (NIC) which is based on the Intel 82599 Ethernet controller. All servers use 3.3 GHz Intel Xeon E3-1230 V2 CPUs. 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.

Fig. 4 shows the setups for tests involving VMs on the DuT. Generating traffic efficiently directly inside a VM proved to be a challenging problem because our load generator is based on the PF_RING DNA framework which only works with certain physical NICs and not with virtual interfaces. Porting it to a VM by using vPF_RING [24], a framework for efficient packet processing in virtual machines, 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 [30]. 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. We therefore assume that the maximum possible throughput 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.

B. Performance Comparison

Table I compares the performance of several forwarding techniques with a single CPU core per VM and switch. DPDK vSwitch is a port of OvS to the user space packet processing framework DPDK [31]. It relies on a modified version of qemu-kvm which proved to be too unstable for tests in our testbed for tests involving VMs. The likely explanation for this instability is that our testbed is based on Debian and DPDK vSwitch is experimental software which is only tested on Fedora [30].

TABLE I. Single Core Data Plane Performance Comparison

Application	pNIC-pNIC [Mpps]	pNIC-vNIC [Mpps]	pNIC-vNIC-pNIC [Mpps]	pNIC-vNIC-vNIC [Mpps]
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	11.31	-	10.5*	6.5*

*) These values were taken from [30]

DPDK vSwitch is the fastest forwarding technique but it is still experimental and not yet ready for real-world use; we include it here to show possible improvements for the next generation of virtual switches. Open vSwitch proves to be the fastest Linux kernel packet forwarding application. The Linux bridge is slightly faster than IP forwarding when it is used as a virtual switch. 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.

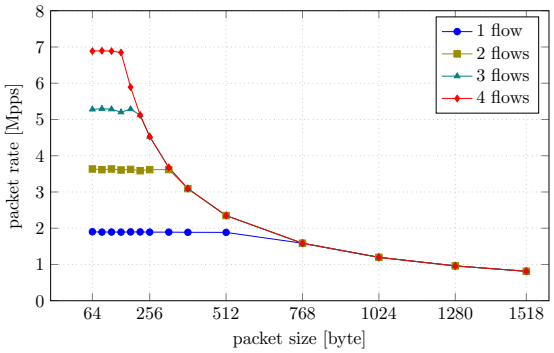
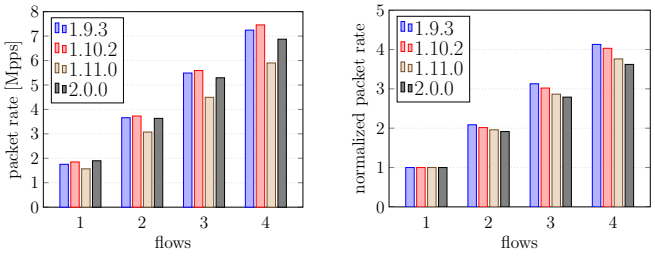


Fig. 5. Throughput with various packet sizes and flows



(a) Packet rates (b) Normalized to one flow

Fig. 6. Throughput of different Open vSwitch versions, 1 to 4 flows

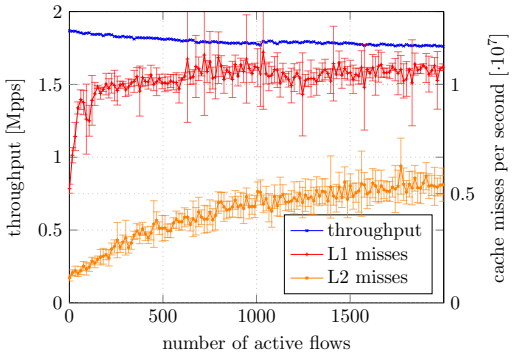


Fig. 7. Flow table entries vs. cache misses

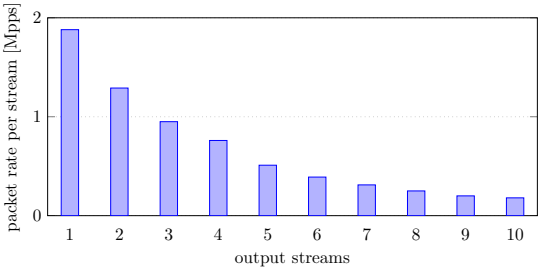


Fig. 8. Effects of cloning a flow

C. Open vSwitch Throughput with pNIC to pNIC Forwarding

Fig. 5 shows the basic performance characteristics of OvS in an unidirectional forwarding scenario between two physical NICs 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. Fig. 6 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 while 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 wild carded 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 would not exhibit this behavior. Linear scaling with the number of CPU cores can therefore be assumed and further tests are restricted to a single CPU core. The throughput per core is 1.88 Mpps.

We derive a test case from the OvS architecture described in Section II-B: 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.

Fig. 7 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. Fig. 8 shows that performance drops by 30% when a flow is sent out twice and another 25% when it is copied one more time. This shows 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 in a system involving software switches

D. Open vSwitch Throughput with Virtual Network Interfaces

Virtual network interfaces exhibit different performance characteristics from physical interfaces. For example, dropping packets in an overload condition is done efficiently and concurrently in hardware on a pNIC while a vNIC needs to drop

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

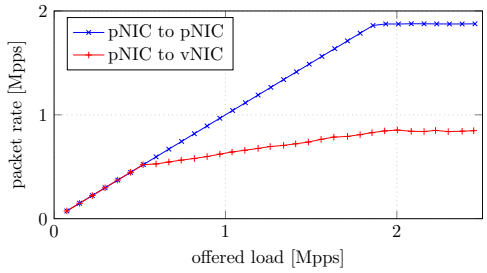


Fig. 9. Offered load vs. throughput with pNICs and vNICs

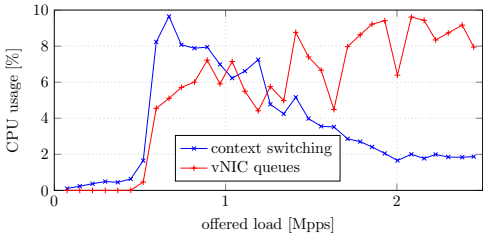


Fig. 10. CPU load of context switching and vNIC queuing

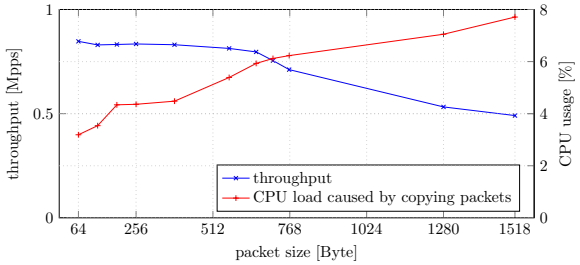


Fig. 11. Packet size vs. throughput and memory copy overhead with vNICs

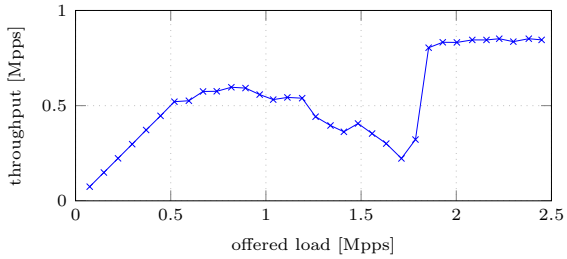


Fig. 12. Throughput without explicitly pinning all tasks to CPU cores

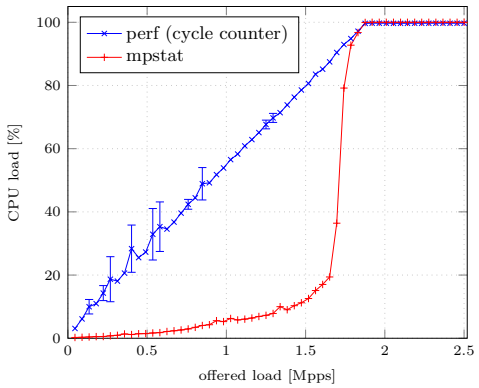


Fig. 13. CPU load of Open vSwitch when forwarding packets

packets in software. We therefore compare the performance of the pNIC to pNIC forwarding with the pNIC to vNIC scenario shown in Fig. 4(a).

Fig. 9 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. Fig. 10 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.

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. Fig. 11 plots the throughput and the CPU load caused by the kernel function `copy_user_enhanced_fast_string`, which copies a packet into the user space, in the forwarding scenario shown in Fig. 4(a). The throughput 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. Fig. 12 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. Fig. 13 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`.

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. The kernel option `CONFIG_IRQ_TIME_ACCOUNTING` can be used to enable accurate reporting of CPU usage by interrupts, this 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.

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 Fig. 4(b) and 4(c) in Section IV-A. 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.

V. 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.

A few guidelines for cloud system operators can be derived from these results: Virtual machines and NIC interrupts should be explicitly pinned to disjoint sets of CPU cores. If pinning all tasks is not feasible, the kernel option `CONFIG_IRQ_TIME_ACCOUNTING` should be enabled which might slow down interrupt handlers (cf. Sec. IV-D). 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.

Virtualized services that rely on bulk data transfer via large packets achieve a high throughput. 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.

Acknowledgments

This research has been supported by the DFG as part of the MEMPHIS project (CA 595/5-2), the KIC EIT ICT Labs on SDN, and the BMBF under EUREKA-Project SASER (01BP12300A).

References

[1] B. Munch, "Hype Cycle for Networking and Communications," Gartner, Report, July 2013.

[2] "OpenStack," https://www.openstack.org/, last visited 2014-05-27.

[3] "OpenNebula," http://opennebula.org/, last visited 2014-05-27.

[4] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the Art of Network Function Virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 459–473.

[5] "Virtual Machine Device Queues: Technical White Paper," Intel Corporation, 2008.

[6] "Intel I/O Acceleration Technology," http://www.intel.com/content/www/us/en/wireless-network/accel-technology.html, Intel Corporation, last visited 2014-05-27.

[7] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting Parallelism To Scale Software Routers," in *22nd ACM Symposium on Operating Systems Principles (SOSP)*, October 2009.

[8] T. Meyer, F. Wohlfart, D. Raumer, B. Wolfinger, and G. Carle, "Validated Model-Based Prediction of Multi-Core Software Router Performance," *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, April 2014.

[9] R. Bolla and R. Bruschi, "Linux Software Router: Data Plane Optimization and Performance Evaluation," *Journal of Networks*, vol. 2, no. 3, pp. 6–17, June 2007.

[10] L. Rizzo, M. Carbone, and G. Catalli, "Transparent Acceleration of Software Packet Forwarding Using Netmap," in *INFOCOM*, A. G. Greenberg and K. Sohraby, Eds. IEEE, 2012, pp. 2471–2479.

[11] "Open vSwitch," http://openvswitch.org, last visited 2014-05-27.

[12] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, August 2000.

[13] L. Deri, "nCap: Wire-speed Packet Capture and Transmission," in *IEEE Workshop on End-to-End Monitoring Techniques and Services*, 2005, pp. 47–55.

[14] L. Rizzo, "Netmap: A Novel Framework for Fast Packet I/O," in *USENIX Annual Technical Conference*, April 2012.

[15] "Impressive Packet Processing Performance Enables Greater Workload Consolidation," Intel Corporation, 2013.

[16] "Intel DPDK: Data Plane Development Kit," http://dpdk.org/, Intel Corporation, last visited 2014-05-27.

[17] L. Rizzo and G. Lettieri, "VALE, a switched ethernet for virtual machines," in *CoNEXT*, C. Barakat, R. Teixeira, K. K. Ramakrishnan, and P. Thiran, Eds. ACM, 2012, pp. 61–72.

[18] K. K. Ram, A. L. Cox, M. Chadha, and S. Rixner, "Hyper-Switch: A Scalable Software Virtual Switching Architecture," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*. San Jose, CA: USENIX, 2013, pp. 13–24.

[19] G. Pongracz, L. Molnar, and Z. L. Kis, "Removing Roadblocks from SDN: OpenFlow Software Switch Performance on Intel DPDK," *Second European Workshop on Software Defined Networks (EWSDN'13)*, pp. 62–67, 2013.

[20] L. Rizzo, M. Carbone, and G. Catalli, "Transparent Acceleration of Software Packet Forwarding using Netmap," in *INFOCOM*. IEEE, 2012, pp. 2471–2479.

[21] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby, "Virtual Switching in an Era of Advanced Edges," in *2nd Workshop on Data Center Converged and Virtual Ethernet Switching (DC-CAVES)*, September 2011.

[22] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending Networking into the Virtualization Layer," in *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[23] M. Dobrescu, K. Argyraki, and S. Ratnasamy, "Toward Predictable Performance in Software Packet-Processing Platforms," in *USENIX Conference on Networked Systems Design and Implementation (NSDI)*, April 2012.

[24] A. Cardigliano, L. Deri, J. Gasparakis, and F. Fusco, "vPFRING: Towards WireSpeed Network Monitoring using Virtual Machines," in *ACM Internet Measurement Conference*, 2011.

[25] G. Wang and T. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *INFOCOM*. IEEE, 2010, pp. 1–9.

[26] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "Openflow switching: Data plane performance," in *International Conference on Communications (ICC)*. IEEE, 2010.

[27] Z. He and G. Liang, "Research and evaluation of network virtualization in cloud computing environment," in *Networking and Distributed Computing (ICNDC)*. IEEE, 2012, pp. 40–44.

[28] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in *Proceedings of the 23rd International Teletraffic Congress*. ITCP, 2011.

[29] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "Oflops: An Open Framework for OpenFlow Switch Evaluation," in *Passive and Active Measurement*. Springer, 2012, pp. 85–95.

[30] "Intel DPDK vSwitch," https://01.org/sites/default/files/page/intel_dpdk_vswitch_performance_figures_0.10.0_0.pdf, Intel Corporation, last visited 2014-05-27.

[31] "Intel DPDK vSwitch," https://github.com/01org/dpdk-ovs, Intel Corporation, last visited 2014-03-25.