Characterizing the Performance of Concurrent Virtualized Network Functions with OVS-DPDK, FD.IO VPP and SR-IOV

Nikolai Pitaev
Enterprise Infrastructure and Solutions Group, Cisco Systems
npitaev@cisco.com

Matthias Falkner
Enterprise Infrastructure and Solutions Group, Cisco Systems
mfalkner@cisco.com

Aris Leivadeas
Department of Systems and Computer Engineering, Carleton University
arisleivadeas@sce.carleton.ca

Ioannis Lambadaris
Department of Systems and Computer Engineering, Carleton University
ioannis@sce.carleton.ca

ABSTRACT

The virtualization of network functions is promising significant cost reductions for network operators. Running multiple network functions on a standard x86 server instead of dedicated appliances can increase the utilization of the underlying hardware,while reducing the maintenance and management costs of such functions. However, total cost of ownership calculations are typically a function of the attainable network throughput, which in a virtualized system is highly dependent on the overall system architecture - in particular the input/output (I/O) path. In this paper we investigate the attainable performance of an x86 host running *multiple* virtualized network functions (VNFs) under different I/O architectures: OVS-DPDK, SR-IOV, and FD.io VPP. Running multiple VNFs in parallel on a standard x86 host is a common use-case for cloud-based networking services. We show that the system throughput in a multi-VNF environment differs significantly from deployments where only a single VNF is running on a server.

CCS CONCEPTS

• **Networks** → *Cloud computing*; *Network servers*; *Network experimentation*; *Network performance analysis*; *Network measurement*; • **Hardware** → *Networking hardware*; Buses and high-speed links;

KEYWORDS

NFV; Virtualized System Architectures; VNF Performance; SR-IOV; OVS; OVS-DPDK; FD.io VPP; Hypervisors; KVM

ACM Reference Format:
Nikolai Pitaev, Matthias Falkner, Aris Leivadeas, and Ioannis Lambadaris. 2018. Characterizing the Performance of Concurrent Virtualized Network Functions with OVS-DPDK, FD.IO VPP and SR-IOV. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering, April 9–13, 2018, Berlin, Germany*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3184407.3184437

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5095-2/18/04…$15.00
https://doi.org/10.1145/3184407.3184437

1 INTRODUCTION

Service Provider and Infrastructure Provider (e.g. Cloud Provider) networks are increasingly making use of virtualized network functions (VNFs) to reap the benefits of reduced capital expenditures (CAPEX) and operating expenses (OPEX). Running VNFs on standard off-the-shelf server platforms promises to reduce the capital expenditures previously dedicated to hardware appliances. Operators are also expecting a significant reduction in the operating expenses by increasing the level of automation enabled by software defined networking. Total cost of ownership calculations however are typically a function of the attainable network performance, which in a virtualized system is highly dependent on the overall system architecture. For input/output (I/O) intensive workloads such as virtualized network functions (VNFs), the packet path from the physical interface on the server into the virtual machine (VM) is particularly impactful on the overall system throughput. Open vSwitch (OVS) [13, 29], OVS - Data Plane Development Kit [8], Single-Root I/O Virtualization [7], and Fast Data input/output Vector Packet Processing (FD.io VPP) [17] are possible alternative mechanisms to carry network traffic from the physical interface into VNFs.

Also impacting the overall system throughput is the number of concurrent VNFs running on top of the hypervisor. For most use-cases where virtualization is considered, multiple VMs are sharing the hardware resources offered by the underlying x86 host. In some cases, application workloads running in VMs are instantiated alongside VNFs. More commonly, operators separate application workloads from VNFs onto different servers, while still instantiating multiple VMs on each of the systems.

The networking departments of many enterprise or service providers operate their own hardware systems, separate from the IT departments. An example of such a deployment scenario is cloud-based managed services. In this use-case, SP operators offer virtualized networking services to enterprise customers out of their own data centers. VNFs such as virtual routers, virtual firewalls, or virtualized WAN optimization are instantiated on a per-enterprise basis. For basic services, a single VNF can be configured to serve multiple enterprise customers, even with multiple networking features (multi-tenancy combined with multi-feature). For more sophisticated services, dedicated VNFs may be service-chained for each enterprise customer, offering higher per-tenant throughput and customer separation. Multiple VNFs associated with different enterprise customers share the underlying x86 host resources in this

case. Sophisticated automation and orchestration environments instantiate such cloud-based networking services within minutes, taking the available hardware resources into account.

Furthermore, multiple VNFs are running in parallel in such a shared server environment, contending for underlying CPU, I/O, memory and storage resources. The overall attainable system throughput may be impacted by various system bottlenecks. In particular, the vSwitch bottleneck is identified as one of the main bottleneck in such a multi-VM configuration. Thus, the performance of different I/O architectures are of utmost importance. In [22], the authors present the performance of running multiple VNFs in parallel while varying the hypervisor's I/O technologies with OVS, SR-IOV and FD.io VPP.

In this paper, we extend the experimental results by replacing OVS with OVS-DPDK, which promises to significantly increase the I/O performance for virtualized network functions. We also use DPDK-enabled VNFs. We show how OVS-DPDK compares from a throughput perspective to SR-IOV and FD.io VPP as the number of VNFs is increased under multiple feature configurations. We demonstrate the system throughput behaviour not just for pure IP forwarding, but also for a realistic virtual router configuration where processing-intensive features like Network Address Translation (NAT), Firewall, Quality-of-Service (QoS) and even deep-packet inspection (DPI) are applied to the packet flows processed by each VNF. Our experiments reveal that OVS-DPDK has comparable performance to FD.io VPP, thus confirming a significant throughput improvement as compared to native OVS. Considering that both OVS-DPDK and FD.io VPP offer richer virtualization functionality with fewer caveats than SR-IOV, network operators looking to deploy VNFs have viable alternatives to trade-off deployment flexibility and overall system throughput.

The remainder of the paper is organized as follows: Section 2 provides an overview of the related work. Section 3 provides insights of the various I/O architectures. Section 4 presents the test methodology followed. The performance evaluation is presented in Section 5. Finally, Section 6 concludes the paper.

2 RELATED WORK

Performance of the I/O path in virtualized system architectures has been widely studied in the literature in recent years. Throughput of FD.io with VPP has been investigated under various system configurations in [19], showing the throughput and latency improvements that can be achieved with FD.io VPP as compared to OVS-DPDK. In [11], the authors perform systematic experiments to investigate various virtual switches, including OVS, Linux bridges and also citing official results for OVS-DPDK. Various traffic flows are tested, going directly into a single VNF, passing through a single VNF, or being service chained through a VNF and terminated in another VNF. A thorough investigation into the system parameters in a similar setup is also reported in [12].

In [20] a user-space virtual switch (SnabbSwitch) is introduced and its performance analyzed against OVS, OVS-DPDK, linux bridges, Virtual Function Input/Output (VFIO), and SR-IOV. The results demonstrate that this implementation performs similar to SR-IOV and VFIO, and out-performing OVS and OVS-DPDK in user-space. The tests are conducted with two VNFs only, generating either uni-directional or bi-directional traffic, following the methodology in [27]. A similar vSwitch development (Lagopus) is described in [23] and its performance studied for both delay and throughput. No comparison to other vSwitches is made, and traffic is not sent to any VNFs.

An elaborate study of SR-IOV performance under multiple VNFs is reported in [10], without making however any comparison to other vSwitches. A similar study for SR-IOV with NAPI optimizations is presented in [15], analyzing CPU utilization and throughput in a multi-VNF scenario, but without comparisons to other I/O techniques. The authors in [24] investigate the switching performance of a virtualized software router with different traffic flows (up to 4) under various packet sizes, and compare the throughput to non-virtualized software routers to show the performance penalty introduced by virtualization, without analyzing different virtual switching technologies. The throughput and latency of OVS and OVS with DPDK with multiple VMs and Docker containers that are service chained is also studied in [2]. The study shows that OVS with DPDK can achieve a 10-fold improvement in the PPS rate for the specified test. However, no comparison to other I/O techniques is made, as the focus of the paper is more on analyzing the number of VMs / containers in a chain. A comparison between physical and VM performance under Intel DPDK is given in [25]. Throughput in terms of Mpps is reported with two VMs and compared to a physical setup. A multi-VNF performance analysis under Openstack is reported in [4], showing also realistic traffic patterns with networking features (DPI, Firewall, routing) applied under Openstack with both a linux bridge and OVS. Newer I/O technologies are not analyzed in the paper.

In comparison with the above presented works, the contribution of this paper is twofold: first, our test study the impact of multiple VNFs running in parallel (not chained) on an x86 host. In such an environment, multiple VNFs place demand for I/O processing and vCPUs on the hypervisor scheduler, thus stressing the latter in a different way to single-VNF tests. This is very common in so-called horizontal-scale use-cases, where multiple VNFs are used in order to fulfil high scale requirements. Our study thus provides insights into a realistic deployment scenario, where there is contention for the servers' hardware resources. In the environment we study, the hypervisor scheduler has to process switch to allocate the available CPU cycles to numerous VNFs as well as to the Linux OS itself. In any of the single-VNF analyses above, such an interaction is not taken into account. Second, we perform our analysis with a commercially available virtual router (the Cisco CSR 1000V® [1]), configured not only for pure IP forwarding, but also showing the impact under a realistic feature processing configuration. Of particular importance in our tests is to avoid any hypervisor tuning steps that may be difficult to operationalize in a production environment. Only those hypervisor or process settings that can be configured at boot time or bring-up are optimized.

3 OVERVIEW OF A VIRTUALIZED SYSTEM ARCHITECTURE

In a basic virtualized system architecture, hardware resources (CPU, Memory, Storage) are being abstracted by a hypervisor layer to present virtual CPU/Memory/Storage to VMs or applications that

run on top of the hypervisor. In this paper, we focus on the scenario of networking functions running inside a VM, as opposed to running VNFs in a container (e.g. Docker [6]) or running applications directly on the host OS. For networking VNFs, one or more software processes are running inside the VM to perform packet processing (e.g. firewall, routing, NAT etc.). These software processes are associated with the virtualized CPUs allocated to the VM. In addition to the vCPU threads configured for a VM, numerous VM system threads are also generating processing loads. The aggregate of all vCPU processes from the set of VMs are presented to the hypervisor layer for scheduling onto physical CPU cores.

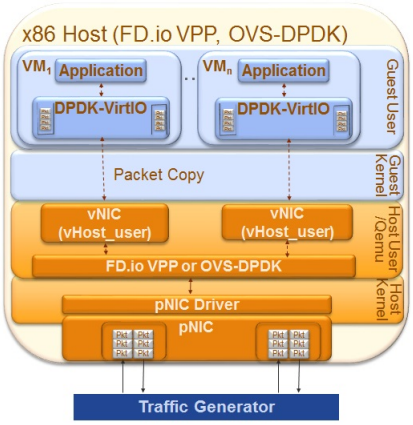
Unfortunately, a virtualized system architecture may expose various throughput bottlenecks. Specifically, the physical port density and speed of the server may constraint the amount of traffic that can be processed. Another bottleneck may be the hypervisor scheduler itself, in particular if a large number of processes need to be allocated CPU cycles with strict timing. Furthermore, the VNF itself typically has a maximum packet processing capacity that may also limit its throughput. Finally, in an I/O bound networking environment another bottleneck is how the packet path from the physical NIC into the VMs can affect the overall performance. In this paper, we are particularly interested in this final bottleneck and for this reason, we evaluate how different available alternatives such as OVS-DPDK, FD.io VPP, or SR-IOV can contribute to the overall system throughput. Our intent is to share real measurement results, and provide insights on the performance of various I/O techniques.

3.1 Virtualized I/O Architectures

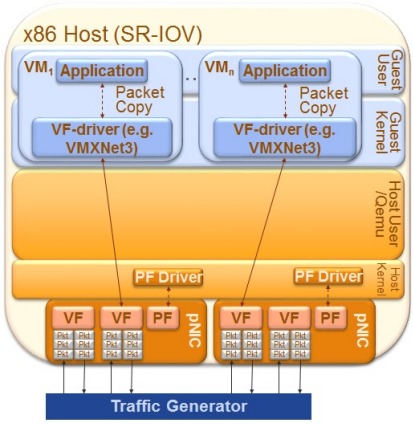
Virtualization of network functions differs from application virtualization. In the former case, the I/O workload generated by packet flows dominates. By definition of a VNF and its purpose being to process networking traffic, packets are continuously arriving into the server and need to be passed to its respective VM for processing. Networking VMs are thus generating high I/O workloads for the hypervisor and thus be referred to as I/O bound. In contrast, many non-networking applications receive only a limited number of external inputs. Their requirement for CPU cycles are predominantly algorithmic computations, possibly also with intensive memory and storage access. Such applications or networking functions consequently become compute-bound.

In general, packets arrive on the physical NIC and are copied into memory via two direct-memory access (DMA) operations. Along with the packet copy, a descriptor specifying the buffer location (memory address and length) is also copied into memory. The pNIC then sends an interrupt to indicate the arrival of the packet (see [10, 24] for details). The packet may then be processed by the virtual switch or a linux bridge process such as OVS-DPDK, FD.io VPP or SR-IOV. The three different system configurations are illustrated in Figures 1a and 1b respectively.

In the case of OVS-DPDK [9], packets are passed to the virtual switch for distribution to the destination VNFs (CSR 1000V® instances), assisted by the data path development kit (DPDK) libraries for fast packet processing. The DPDK libraries offer a poll-mode driver (PMD) that allows packets to pass from the physical interface to the virtual switch directly, thus avoiding the networking



(a) FD.io VPP and OVS-DPDK



(b) SR-IOV

Figure 1: I/O paths of virtualized system architectures

stack of the kernel. OVS-DPDK offers enhanced switching functionality, supporting among others, jumbo-frames, link bonding, native tunnelling support for VXLAN, GRE or Geneve, MPLS, or ingress/egress policing. From a CPU resource perspective, OVS-DPDK is relying on CPU cycles from the host x86 core to switch packets, thus stressing the hypervisor scheduler in a system where multiple VNFs are also contending for the same CPU cycles. Any CPU core associated for switching to OVS-DPDK becomes unavailable to process VNFs. OVS-DPDK can however be configured to use multiple CPU cores for packet switching to increase its throughput towards the VNFs. In our tests below, we have configured 2 cores to be used for switching packets. Note that Figure 1a highlights the pertinent queues in these setups. Such internal queues are setup to pass packets on their path from the virtual switch into the VNFs, and their depths can become a bottleneck with high data rates. For OVS-DPDK the pertinent queues are in the DPDK driver in the guest user space. Note also that the VNFs used in our tests (the Cisco CSR 1000v) are also supporting DPDK to transfer packets efficiently to and from OVS-DPDK, so the entire packet path from the physical interface via the virtual switch into the VNF is supported by DPDK.

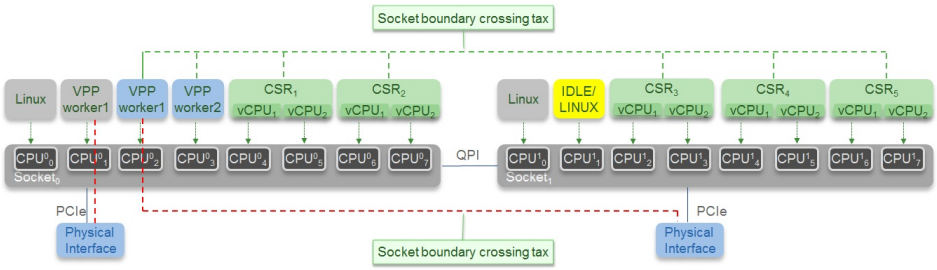


Figure 2: 2-socket x86 NUMA architecture

FD.io VPP [17] is an open-source alternative solution to optimize the I/O path in a virtualized system. Running as a Linux user-space process, the FD.io VPP drivers enable NIC access over PCI. FD.io processes packets in vectors (vector packet processing, VPP). Packets are removed from the receive rings of the interface and are formed into a packet vector, to which a processing graph is then applied [14]. The processing graph represents the features that need to be applied (e.g. IPv4 forwarding, classification, multicast etc.) [17]. This approach minimizes interrupts and traversing a call stack and thus also thrashing of the instruction caches and misses. VPP processes multiple packets at a time, making it a high-performance processing stack that supports even more networking functions than OVS-DPDK. Features such as DHCP, Segment routing, ARP, L2TPv3, VRFs, IPv6, MPLS-over-Ethernet are all supported. Similar to OVS-DPDK, FD.io VPP makes use of Intel's DPDK [8] library to accelerate packet processing, and thus requires CPU cycles to process packets which become unavailable for VNF processing. Again, the number of CPU cores assigned to FD.io VPP can be configured, and is set to 2 in our test to align with the OVS-DPDK setup. FD.io VPP also leverages internal queues in the DPDK driver in the guest user space to pass packets from the virtual switch into the VNFs.

Figure 2 provides a sample illustration of the hypervisor's I/O configuration for FD.io VPP (and can be generalized for OVS-DPDK). The x86 server is shown to offer two sockets with 8 cores each in the diagram. The PCI links of the physical interface are associated with the first socket, which also runs the switching threads (worker-threads) for FD.io VPP. The VNFs are pinned to dedicated cores spread across both sockets. The non-uniform memory access (NUMA) [16] architecture provides separate memory for each processor core and thus enables the cores of hitting their respective memory banks in parallel.

SR-IOV [7] in contrast offers a virtualized PCIe pass-through mechanism that does not rely on the hypervisor to pass packets between the NIC to the individual VNFs. As illustrated in Figure 1b SR-IOV virtualizes PCIe, creating PCIe physical functions (PF) and virtual functions (VF). This allows a physical port to be shared amongst multiple VNFs. The processing of features in a SR-IOV setup is entirely done inside the VNF, requiring the VNF to support the appropriate drivers. Features such as VXLAN, MPLS, policing etc. mentioned above for OVS-DPDK and FD.io VPP now have to be applied to packets inside the VNFs. SR-IOV has some functional limitations [28] due to its dependency on the underlying hardware and software. The server's NIC cards and the BIOS have to support the technology. Further caveats are for example the number of VFs that can be configured for a physical NIC, currently limiting the number of VFs to 128 on an Intel Fortville NIC - but the practical limit may be as low as 64 [28]. Depending on the hardware and the driver implementation, other caveats may exist such as packet mirroring, VLAN filtering, multicast addresses or promiscuous unicast [26].

4 TEST METHODOLOGY

To demonstrate the effects of the various I/O options in a multi-VNF system configuration, we apply the following test methodology: A Cisco UCS C240 Series [5] x86 host configured with a Redhat KVM [18] virtualization infrastructure is connected via a NIC with two 10 Gigabit Ethernet (GE) interfaces to a Layer 2 switch (Cisco Nexus 5548), which is in turn connected to an IXIA traffic generator using again 2x10 GE ports. The traffic generator sends IP Traffic with either an IMIX, 128B or 1518B packet size to a variable number of Cisco CSR 1000V® virtual routers hosted on the x86 server. IMIX traffic refers to typical internet traffic with packet sizes distributed within the range of 64 to 1500B. Up to 10 virtual routers are instantiated to ensure that the physical hardware resources are not over-subscribed.

The Cisco 1000V VNFs are configured to either perform basic IPv4 forwarding on the traffic stream (IP Throughput test), or to apply feature processing by executing Network Address Translation (NAT), Firewall, Quality-of-Service (QoS) and Deep-Packet Inspection (DPI) on the traffic stream. The latter feature configuration is representative for the cloud-based managed customer premises equipment (CPE) use-case described in Section 1. The bi-directional traffic is then returned to the traffic generator to measure the attained throughput, packet loss and other statistics, following RFC2544 [3], accepting a packet loss rate of 0.01% over the 1 minute measurement interval[1]. The specifics of the test setup are summarized in Table 1.

The packet path from the physical port in the x86 host to the VNFs is compared for three different system configurations, illustrated in Figures 1a, 1b for SR-IOV and FD.io VPP, and OVS-DPDK respectively, as described in Section 3.

[1]In virtualized systems, a non-zero packet loss rate is typically accepted since the Linux scheduler is not designed to accommodate a no drop rate (NDR) for high packet I/O.

Table 1: Details of the Test components

Test Component	Details
x86 Host	Cisco UCS C240 M4 Series: 2 Sockets Intel Xeon E5-2699v3 2.3 GHz with 18 cores each, 262GB RAM
Physical Interfaces	1 NIC with 2 x 10GE ports; Intel X520-DA2 NIC
Hypervisor	Redhat KVM version 7.2; Linux kernel 3.10.0-327.18.2.el7.x86 64; Libvirt 1.2.17; QEMU version 2.3.0
I/O Paths	OVS version 2.4.0
	Cisco FD.io VPP release 16.06, configured for 3 cores
	SR-IOV
Switch	Cisco Nexus 5548 Series, NX-OS 7.0
Traffic Generator	Ixia N2X, IxServer6.80.1100.12 GA
VNFs	Cisco CSR 1000V® virtual Router, IOS XE version 16.3.1a; 2 vCPU, 4GB RAM

5 MULTI-VM PERFORMANCE RESULTS

In this section we present the performance of the different I/O architectures in a multi-VM system. Two separate experiments were carried out. Experiment 1 with the goal to evaluate the performance of the throughput achieved when VNFs are configured to basic IPv4 forwarding (Cisco Express Forwarding - CEF); and Experiment 2 with the goal to evaluate the throughput performance achieved when applying feature processing by executing the feature set (NAT, Firewall, QoS, and DPI) to the VNF.

In our test methodology, the measured results are compared to a benchmark throughput that represents the optimal multi-VM throughput. The benchmark is derived from the measurement of a single VNF test with SR-IOV with either the IP forwarding only (Cisco Express Forwarding - CEF) or the feature set NAT, Firewall, QoS and DPI applied to the VNF. In ideal conditions, each additional VNF would contribute the same throughput to the overall system performance as the first VNF, up to the point where other system bottlenecks (in particular the physical NIC capacity) are reached. The benchmark is thus an additive linear extrapolation of the single-VNF test under SR-IOV. In other words, it is a theoretical optimal increase of the throughput that we would expect to notice as we add VNFs on the server. Details on a similar test setup are reported in [21].

5.1 Experiment 1: IPv4 Forwarding Results

Figures 3 and 4 show the measured performance with multiple VNFs for pure IP forwarding with an IMIX packet size in terms of Gbps and MPPS respectively[2]. SR-IOV shows a near-linear contribution to the overall system throughput for each additional VNF, reaching the physical interface limit of 19.01 Gbps with 3 VNFs under IMIX. The discrepancy between the full interface bandwidth of 20 Gbps and the measured maximum rate is explained by accounting for the inter-frame gap, the preamble and the start-of-frame delimiter in the Ethernet header. The maximum expected packet forwarding rate across both 10GE interfaces is 6.18 MPPS. The observed forwarding rate with SR-IOV of 6.14 is almost matching this rate.

In the case of FD.io VPP and OVS-DPDK, additional VNFs contribute positively to the overall system throughput for the first 2 and 3 VMs respectivey, reaching beyond a system throughput of

[2]Results for a packet size of 1518 bytes are not shown for the IP forwarding traffic profile. Both SR-IOV and FD.io VPP are able to exhaust the physical interface capacity already with a single VNF. OVS-DPDK is able to reach the physical interface capacity with two VNFs.

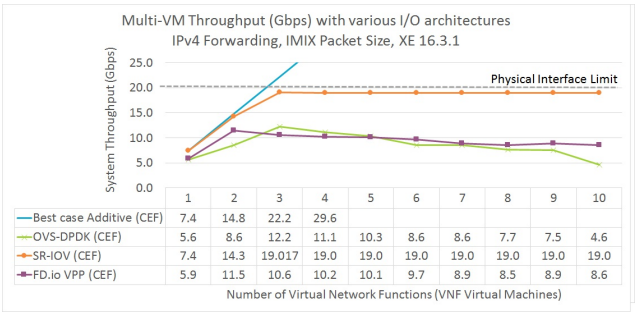


Figure 3: Multi-VM System Throughput with CEF, IMIX Packet Size, Gbps

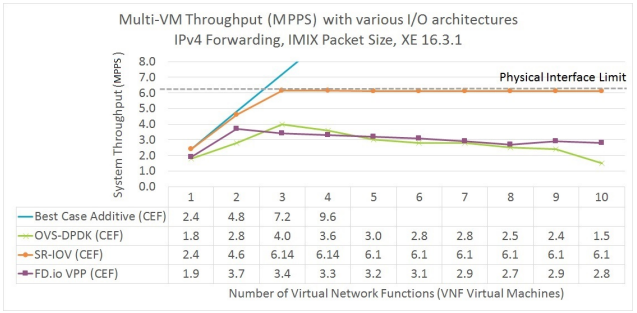
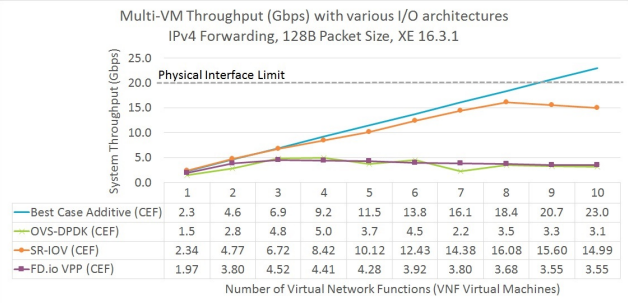


Figure 4: Multi-VM System Throughput with CEF, IMIX Packet Size, MPPS

10 Gbps. However, additional VNFs instantiated thereafter cause the system throughput to decline. This decline is slightly more significant for OVS-DPDK than for FD.io VPP. Such a decline implies a decreasing *average* throughput per VNF as VNFs are added (system throughput divided by number of VNFs), challenging a network operators capacity planning rules. Neither of the latter two I/O techniques reaches the physical interface limit of 20 Gbps in this test. The gradual decline in system throughput is further investigated below.

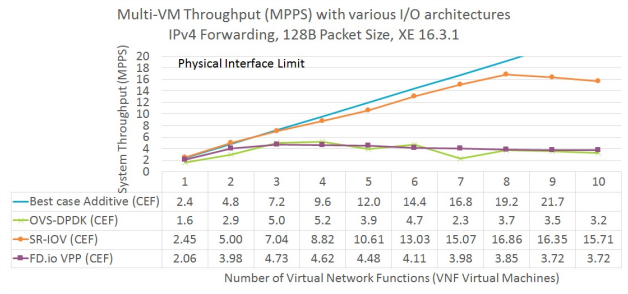
As expected, the total system throughput is scaled proportionally with a smaller packet size of 128B (see Figures 5 and 6), causing the slope of the benchmark to be shallower. Again, SR-IOV shows a linearity in system throughput as the number of VNFs is increased, up to 8 VNFs. Thereafter however, additional VNFs no longer contribute positively to the overall system throughput, and the maximum interface capacity of 20 Gbps cannot be reached even with 10 instantiated VNFs. Taking the Ethernet overhead (interframe gap, preamble) into account explains the upper limit. To account for the subsequent degradation of system throughput with the 9th and 10th VNF, we examined the packet loss counters of the physical interfaces. We observed packet losses towards the traffic generator, indicating that the NIC buffers are overflowing.

OVS-DPDK and FD.io VPP as I/O mechanisms for 128B packet flows again reach their maximum system throughput with 3 VNFs at around 5 Gbps and 5 Mpps respectively. The total system throughput again tapers off as additional VNFs are added. It is notable that



	1	2	3	4	5	6	7	8	9	10
Best Case Additive (CEF)	2.3	4.6	6.9	9.2	11.5	13.8	16.1	18.4	20.7	23.0
OVS-DPDK (CEF)	1.5	2.8	4.8	5.0	3.7	4.5	2.2	3.5	3.3	3.1
SR-IOV (CEF)	2.34	4.77	6.72	8.42	10.12	12.43	14.38	16.08	15.60	14.99
FD.io VPP (CEF)	1.97	3.80	4.52	4.41	4.28	3.92	3.80	3.68	3.55	3.55

Figure 5: Multi-VM System Throughput with CEF, 128B Packet Size, Gbps



	1	2	3	4	5	6	7	8	9	10
Best case Additive (CEF)	2.4	4.8	7.2	9.6	12.0	14.4	16.8	19.2	21.7	
OVS-DPDK (CEF)	1.6	2.9	5.0	5.2	3.9	4.7	2.3	3.7	3.5	3.2
SR-IOV (CEF)	2.45	5.00	7.04	8.82	10.61	13.03	15.07	16.86	16.35	15.71
FD.io VPP (CEF)	2.06	3.98	4.73	4.62	4.48	4.11	3.98	3.85	3.72	3.72

Figure 6: Multi-VM System Throughput with CEF, 128B Packet Size, MPPS

for small packet sizes of 128B, the attainable system throughput of both FD.io VPP and OVS-DPDK is very comparable.

Next, we examine the impact of varying the switching threads (VPP worker threads for FD.io) on the total system throughput. Figure 7 shows the detailed CPU core configurations with 1, 2 and 4 FD.io VPP switching threads respectively[3]. Multiple VNF instances were again instantiated on the x86 host and subjected to IPv4 traffic with IMIX. Figure 8 shows the corresponding results for this separate experiment. With a single switching thread pinned to a dedicated core, the throughput level reaches 6.31 Gbps with a single VNF, and then again exhibits a slight degradation as VNFs are added to the system. Configuring a second switching thread to the configuration increases the overall system throughput to over 8 Gbps, and then sees the system throughput level off at around 9 Gbps. Configuring further two switching threads (for a total of 4) to the FD.io VPP configuration significantly increases the system throughput. However, in this case the CPU resources are exhausted after instantiating 4 VNFs, demonstrating the trade-off of allocating CPU cores to VNFs or to the I/O path.

A more thorough investigation of the results also shows that the association between the physical interfaces, the switching threads, and the VNFs onto the sockets can impact performance. Instantiating VNFs on the one socket while the physical interface and the FD.io VPP switching thread are pinned to the other socket (c.f. Figure 2) forces remote memory accesses across sockets (a socket boundary crossing across the QPI link). In a NUMA architecture, this causes a slight performance degradation and is at least a partial explanation for the slight degradation of the system throughput observed in Figures 3 or 5. This socket boundary crossing penalty is more pronounced if the physical interfaces' PCI is associated with different socket to where the FD.io VPP switching threads are processed. Placing VNFs on different sockets to the FD.io VPP switching thread incurs less of a penalty.

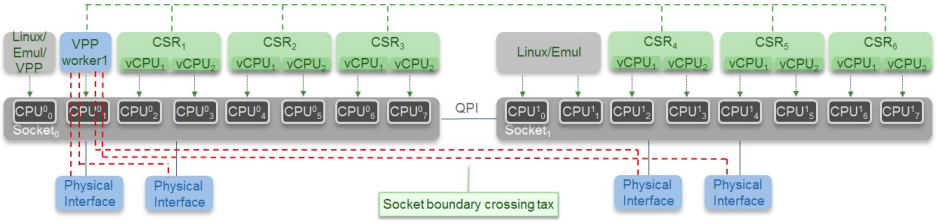
[3]These test were run on a Cisco UCS C-Series C240M4SX with two Intel E5-2667v3 sockets, 8 cores each, clocked at 3.2 GHZ. Traffic arriving on 4x10GE interfaces, two associated with each socket. The hypervisor details were consistent as in Table 1. The IOS XE version tested was 16.3.0, accounting for some differences in the measured throughput results.

5.2 Experiment 2: Throughput with features

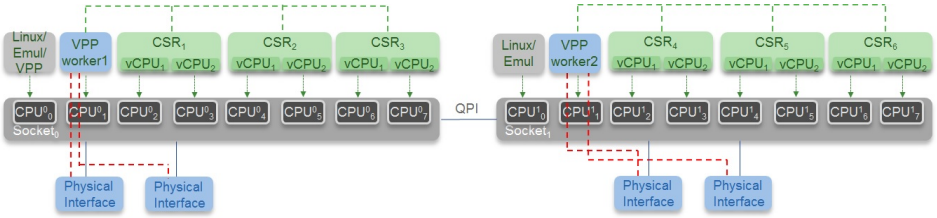
The performance profile observed for IP traffic flows changes considerably as features are turned on in the VNFs. More processing cycles are required by the VNFs themselves to manipulate the traffic, in our case applying DPI, NAT, QoS, and Firewall. The overall system throughput that can be achieved in such a configuration is consequently below the IP forwarding throughput, exhibited in Figure 9 by having a significantly smaller gradients as VNFs are added (benchmark comparison between Figures 3 and 9). Interestingly, in this case the system throughput for SR-IOV and FD.io VPP is almost identical as VNFs are added, up to 6 VNFs, and also highly linear, demonstrating the performance of the vector processing approach with FD.io VPP. This linearity is desirable from an operator perspective since it provides the necessary determinism for capacity planning as customers for the networking services are added. Note that for FD.io VPP the system throughput increases up to 6 VNFs and aggregates 9 Gbps. This is below the maximum throughput of 10Gbps observed above for FD.io VPP under IMIX - the feature processing overhead of the VNFs shifts the processing burden to the VNFs instead of the I/O path, and so the switching capacity limits of the FD.io VPP worker threads are not stressed. As observed for the IMIX tests, the system throughput starts to degrade from the 8th VNF onwards. The socket boundary crossing architecture described above offers an explanation for this degradation: for VNFs 7-10 the switching threads are on a different socket, thus forcing a boundary crossing across the QPI link.

For the multi-feature test case, OVS-DPDK exhibits near-linearity until 5 VNFs, but trailing noticeably below SR-IOV and FD.io VPP. However, after the 6th VNF the throughput degradation under OVS-DPDK is significant. Recall that this implies impacts on the already instantiated VNFs as new VNFs are added - the *average* throughput per VNF is 390Mbps for the 10VNFs, so less than half of the throughput achieved for a single VNF (900 Mbps). The NUMA architecture with socket boundary crossings across the QPI link explains part of this degradation. Again, VNFs 7-10 are associated with a different socket than the OVS-DPDK PMD threads, thus forcing cross-socket memory access. However, since the system throughput degradation starts after the 6th VNF, it means that a limit of the OVS-DPDK switching capacity is also reached.

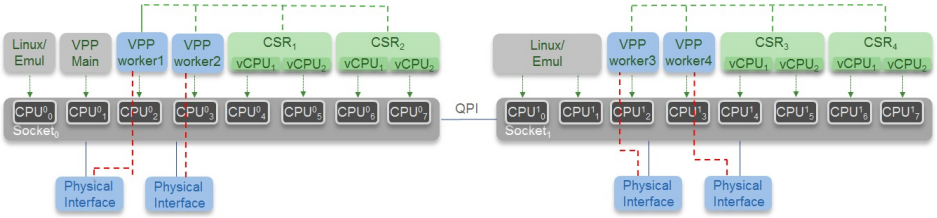
The measurements of the system throughput with large packet sizes (1500B) and with features enabled is shown in Figure 10. In this case, the larger packet size implies that fewer packets are required to fill the physical interface bandwidth, and so no bottlenecks in the I/O path are experienced for the SR-IOV and FD.io VPP cases. The physical interface bandwidth limit of 20Gbps is reached after



(a) 1 switching thread

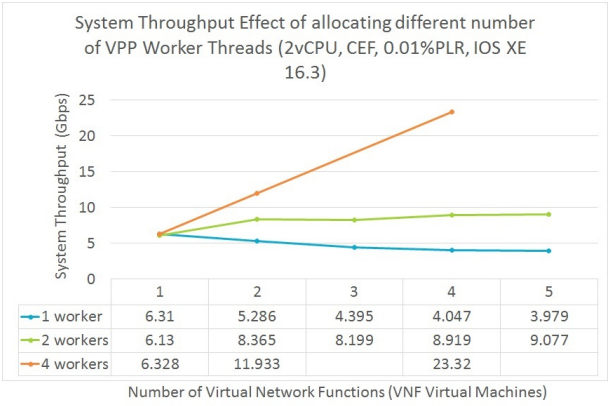


(b) 2 switching threads



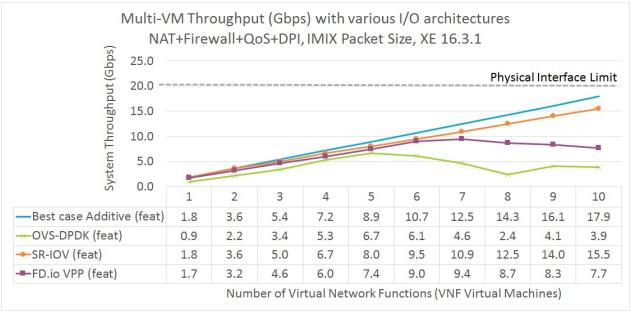
(c) 4 switching threads

Figure 7: System Architecture for different FD.io VPP switching thread experiments



	1	2	3	4	5
1 worker	6.31	5.286	4.395	4.047	3.979
2 workers	6.13	8.365	8.199	8.919	9.077
4 workers	6.328	11.933		23.32	

Figure 8: Impact of different worker threads on system throughput



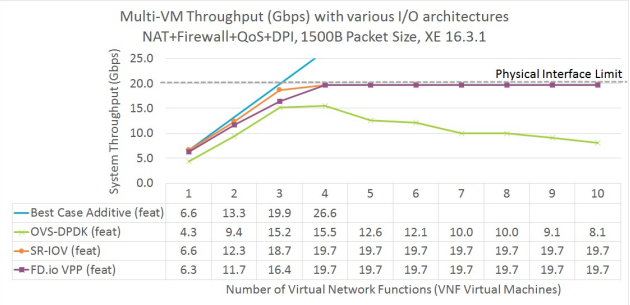
	1	2	3	4	5	6	7	8	9	10
Best case Additive (feat)	1.8	3.6	5.4	7.2	8.9	10.7	12.5	14.3	16.1	17.9
OVS-DPDK (feat)	0.9	2.2	3.4	5.3	6.7	6.1	4.6	2.4	4.1	3.9
SR-IOV (feat)	1.8	3.6	5.0	6.7	8.0	9.5	10.9	12.5	14.0	15.5
FD.io VPP (feat)	1.7	3.2	4.6	6.0	7.4	9.0	9.4	8.7	8.3	7.7

Figure 9: Multi-VM System Throughput with Features, IMIX Packet Size, Gbps

instantiation of the 4th VNF. An I/O path based on OVS-DPDK initially also shows linearity up to the 3rd VNF, reaching 15.2Gbps per server. Thereafter, the total system throughput flattens and starts to decline significantly again after the 4th VNF, pointing again to the less efficient packet processing paradigm of OVS-DPDK as compared to FD.io VPP. Adding more PMD threads to the OVS-DPDK configuration can improve system throughput at the expense of instantiating fewer VNFs.

6 CONCLUSION AND FUTURE WORK

In this paper we analyzed the performance of running multiple VNFs in parallel under different I/O technologies: SR-IOV, OVS-DPDK and FD.io VPP. Our tests differ from service chained deployments in that each VNF is executing all the functions required for



	1	2	3	4	5	6	7	8	9	10
Best Case Additive (feat)	6.6	13.3	19.9	26.6						
OVS-DPDK (feat)	4.3	9.4	15.2	15.5	12.6	12.1	10.0	10.0	9.1	8.1
SR-IOV (feat)	6.6	12.3	18.7	19.7	19.7	19.7	19.7	19.7	19.7	19.7
FD.io VPP (feat)	6.3	11.7	16.4	19.7	19.7	19.7	19.7	19.7	19.7	19.7

Figure 10: Multi-VM System Throughput with Features, 1500B Packet Size, Gbps

a particular IP traffic flow, instead of service chaining the traffic through multiple VNFs. The hypervisor scheduler thus not only has to fulfil the I/O demands of the VNFs, but also the vCPU demands arising in such a scenario. Our studies show that SR-IOV offers a highly linear contribution to the total system throughput as VNFs are added to a server. However, SR-IOV also comes with some deployment caveats that may restrict a deployment scenario.

FD.IO VPP and OVS-DPDK perform very similar in our test environments, scaling linearly for the initial VNFs, but then reaching a system throughput plateau as further VNFs are added. We show that this plateau is a function of the CPU resources allocated to the virtual switching functions of FD.io VPP and OVS-DPDK. Both FD.io VPP and OVS-DPDK demonstrate a gradual decline in total system throughput as further VNFs are added. We investigated this decline and identified the pinning configuration of the physical interfaces, the virtual switching threads, as well as the VNFs as contributing to this gradual decline. In our tests with demanding features (DPI, NAT, Firewall, QoS), FD.IO VPP outperformed OVS-DPDK, showing the benefits of processing packets in a vector. Both of these I/O techniques are thus viable alternatives to SR-IOV that offer better feature functionality and deployment flexibility if the deployment scenario justifies the resource profiles required.

As further research areas, we are identifying to run similar tests in an oversubscribed setup, where the sum of the offered vCPU resources from the VNFs exceeds the available core count on the x86 host. Such measurements allow a better characterization of the hypervisor scheduler under heavy loads, and are also relevant for SPs and Cloud Providers to further increase the VNF density per server and thus reduce the overall total cost of ownership.

REFERENCES

[1] [n. d.]. Cisco cloud services router 1000v series. ([n. d.]). Retrieved October 13, 2017 from http://www.cisco.com/c/en/us/products/routers/cloud-services-router-1000v-series/index.html

[2] R. Bonafiglia, I. Cerrato, F. Ciaccia, M. Nemirovsky, and F. Risso. 2015. Assessing the Performance of Virtualization Technologies for NFV: a Preliminary Benchmarking. In *Proc. of the 4th IEEE Eur. Wrkshp on Software Defined Networks (EWSDN)*, IEEE (Ed.). 67–72. https://doi.org/10.1109/EWSDN.2015.63

[3] S. Bradner and J. McQuaid. 1999. *Benchmarking Methodology for Network Interconnect Devices.* Technical Report RFC2544. Internet Engineering Task Force (IETF).

[4] F. Callegati, W. Cerroni, and C. Contoli. 2016. Virtual Networking Performance in Openstack Platform for Network Function Virtualization. *J. of Electrical and Computer Engineering* 2016 (March 2016), 1–15. https://doi.org/10.1155/2016/5249421

[5] Cisco. [n. d.]. Cisco ucs c240 m4 rack server. ([n. d.]). Retrieved October 13, 2017 from http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-c240-m4-rack-server/index.html

[6] Docker Container. [n. d.]. ([n. d.]). Retrieved October 10, 2017 from https://www.docker.com/

[7] Intel Corporation. 2011. Pci-sig SR-IOV Primer: An Introduction to SR-IOV Technology. (2011). Retrieved October 10, 2017 from http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html

[8] Intel Corporation. 2014. Intel DPDK vSwitch: Performance Report. (2014). Retrieved October 10, 2017 from https://01.org/sites/default/files/page/intel_dpdk_vswitch_performance_figures_0.10.0_0.pdf

[9] Intel Corporation. 2016. Open vSwitch* with DPDK Overview. (2016). Retrieved October 13, 2017 from https://software.intel.com/en-us/articles/open-vswitch-with-dpdk-overview

[10] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan. 2016. High Performance Network Virtualization with SR-IOV. In *Proc. of the 16th IEEE Int. Symp. on High Performance Computer Architecture (HPCA)*, IEEE (Ed.). 1–10. https://doi.org/10.1109/HPCA.2010.5416637

[11] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle. 2014. Performance Characteristics of Virtual Switching. In *Proc. of the 3rd IEEE Int. Conf. on Cloud Networking (CloudNet)*, IEEE (Ed.). 120–125. https://doi.org/10.1109/CloudNet.2014.6968979

[12] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle. 2015. Assessing Soft and Hardware Bottlenecks in Pc-based Packet Forwarding Systems. In *Proc. of the 14th Int. Conf. on Networks (ICN)*. 78–83.

[13] B Pfaff et al. 2015. The Design and Implementation of Open vSwitch. In *In Proc. of the 12th USENIX Symp. on Networked Systems Design and Implementation*, USENIX (Ed.). 117–130.

[14] FD.io. 2016. Fd.io /dev/boot. (2016). Retrieved October 13, 2017 from https://docs.google.com/presentation/d/1JL5O_ZkRUXVaY4ZuKaMj13jEGv90eanOYAB7mHfmPK4/pub?start=false&loop=false&delayms=3000#slide=id.p4

[15] Z. Huang, J. Li, Z. Chang, and H. Guan. 2012. Adaptive and Scalable Optimizations for High Performance SR-IOV. In *Proc. of the IEEE Int. Conf. on Cluster Computing*, IEEE (Ed.). 459–467. https://doi.org/10.1109/CLUSTER.2012.28

[16] C. Kim and K. Park. 2015. Credit-based Runtime Placement of Virtual Machines on a Single NUMA System for QoS of data access performance. *IEEE Trans. on Computers* 64, 6 (June 2015), 1633–1646. https://doi.org/10.1109/TC.2014.2329671

[17] M. Konstantynowicz. [n. d.]. FD.io - How to Push Extreme Limits of Performance and Scale with Vector Packet Processing Technology. ([n. d.]). Retrieved October 10, 2017 from https://www.ietf.org/proceedings/96/slides/slides-96-bmwg-10.pdf

[18] KVM. [n. d.]. Kernel virtual machine. ([n. d.]). Retrieved October 13, 2017 from https://www.linux-kvm.org/

[19] Lightreading Ray Le Maistre. 2015. Validating Cisco's NFV Infrastructure Part 1. (2015). Retrieved October 13, 2017 from http://www.lightreading.com/nfv/nfv-tests-and-trials/validating-ciscos-nfv-infrastructure-pt-1/d/d-id/718684

[20] M. Paolino, N. Nikolaev, J. Fanguede, and D. Raho. 2015. SnabbSwitch User Space Virtual Switch Benchmark and Performance Optimization for NFV. In *Proc. of the IEEE Conf. on Network Function Virtualization and Software Defined Networking (NFV-SDN)*, IEEE (Ed.). 86–92. https://doi.org/10.1109/NFV-SDN.2015.7387411

[21] N. Pitaev. [n. d.]. Cisco CSR 1000v Multi VM / Multi IO Test Report. ([n. d.]). Availableupondemand

[22] N. Pitaev, M. Falkner, A. Leivadeas, and I. Lambadaris. 2017. Multi-VNF Performance Characterization for Virtualized Network Functions. In *Proc. of the IEEE Conf. on Network Softwarization (Netsoft)*, IEEE (Ed.). 1–5. https://doi.org/10.1109/NETSOFT.2017.8004221

[23] R. Rahimi, M. Veeraraghavan, Y. Hakajima, H. Takahashi, S. Okamoto, and N. Yamanaka. 2016. A High-Performance Openflow Software Switch. In *Proc. of the 17th IEEE Int. Conf. on High Performance Switching and Routing (HPSR)*, IEEE (Ed.). 93–99. https://doi.org/10.1109/HPSR.2016.7525645

[24] R. Rojas-Cessa, K. Salehin, and K. Egoh. 2015. Evaluation of Switching Performance of a Virtual Software Router. In *Proc. of the 35th IEEE Sarnoff Symp.*, IEEE (Ed.). 1–5. https://doi.org/10.1109/SARNOF.2012.6222733

[25] V. Sankaran and D. Darde. 2015. Performance Analysis of Intel DPDK on Physical and Virtual Machines. (2015). Retrieved October 13, 2017 from http://www.cs.cornell.edu/courses/cs5413/2014fa/projects/group_of_dsd96_vs444/final_pres.pdf

[26] H. Shimamoto. 2016. SR-IOV ixgbe driver limitations and improvement. (2016). Retrieved October 13, 2017 from http://events.linuxfoundation.org/sites/events/files/slides/20160715_LinuxCon_sriov_final.pdf

[27] M. A. Tahhan and J. M. Morgan. [n. d.]. Vsperf Deep Dive: Virtual Switch Performance in OPNFV. ([n. d.]). Retrieved October 13, 2017 from https://wiki.opnfv.org/display/vsperf/?preview=/2926262/6818343/VSPERF%20Golden.pptx

[28] VMware. [n. d.]. VMware vSphere 5.1 Documentation Center. SR-IOV Support. ([n. d.]). Retrieved October 13, 2017 from https://pubs.vmware.com/vsphere-51/index.jsp?topic=%2Fcom.vmware.vsphere.networking.doc%2FGUID-E8E8D7B2-FE67-4B4F-921F-C3D6D7223869.html

[29] Open vSwitch. [n. d.]. ([n. d.]). Retrieved October 10, 2017 from http://openvswitch.org