TECHNICAL BRIEF

Implementing a TCP Broadband Speed Test in the Cloud for Use in an NFV Infrastructure

When communication service providers consider virtualizing their network architectures, one of the key components of focus is TCP performance, which is fundamental for most networking services.

Muhammad A. Siddiqui
Solution Engineer

Przemysław Lal
Solution Engineer

Tarek Radi
Lead Solution Enabling Manager

Mark Kavanagh
Application Engineer

Introduction

Since 2015, Intel Corporation has cooperated with several communication service providers (Comms SPs) to define optimal network functions virtualization infrastructures (NFVIs) that would help virtualize their workloads.

As part of this initiative, Intel helped a prominent video, phone, and cable Internet access provider to build a performant generic infrastructure that could be used for different workloads such as video on demand and high-speed, data-related network functions. This infrastructure had to be capable of supporting edge and access services across technologies such as Data Over Cable Service Interface Specifications (DOCSIS), Ethernet, or passive optical networks.

Many Internet customers use a speed test server as a tool to compare the actual speed they are experiencing with the speed they signed up for. These servers are based on the transport control protocol (TCP); therefore, TCP performance is critical in such a network infrastructure.

At the time of this writing, there was no publicly available speed test server that could run in a virtualized setup and achieve download and upload rates greater than 1 Gbps using a single virtual machine. Hence, to define the ideal TCP tunings necessary to deploy a performant NFVI, we relied on iPerf3* instead.

This document describes the hardware and software components as well as TCP performance optimizations implemented in order to deliver an optimal NFVI capable of handling communications-grade network functions virtualization (NFV) workloads. The document also briefly discusses three key test scenarios that reflect real-world workloads. We share performance test results for these three scenarios and summarize all of our key findings. Our findings will benefit readers who are considering virtualizing their topology and are more interested in TCP than User Datagram Protocol (UDP) workloads, regardless of whether they plan to deploy a speed test server.

Our research shows that open source software can be successfully used to achieve a reliable and flexible topology that enables delivery of robust and highly efficient virtualized infrastructures. We present an example of such an infrastructure that is built upon standard servers with Intel® Xeon® processors and is optimized to set up a virtualized broadband speed test server. The infrastructure is capable of processing external workloads with an average throughput of **9.35 Gbps** over a 10 Gbps network connection and internal workloads with a throughput reaching **45 Gbps**.

An Open Source Infrastructure

The NFVI used for this proof-of-concept (PoC) consists of three commercial off the-shelf (COTS) Intel® Xeon® processor-based servers (see Table 2 in the Appendix), running the Fedora* 21 Server operating system. We installed OpenStack* Kilo on these three servers. One server was configured as an OpenStack controller that also includes the OpenStack Networking* functions, whereas the remaining two servers were configured as compute nodes. Figure 1 shows the physical topology and software stack used in this PoC.

OpenStack is a well-known open source software platform that enables users to deploy, orchestrate, and manage virtual network functions (VNFs) in the cloud using a single web-based dashboard. New NFV extensions integrated into OpenStack Kilo include support for non-uniform memory access (NUMA) architectures, CPU affinity in virtual machines (VM), and huge pages.

These extensions enabled us to select particular CPU cores, memory banks, and network interface cards (NICs) that would belong to the same NUMA node, and dedicate these resources to specific performance-demanding VMs, such as the broadband speed test server.

Open vSwitch* (OvS*), is a production-quality, widely deployed open source virtual software switch. OvS's primary switching component is a *"fastpath"* kernel module, which is programmed and controlled by user-space processes. This fastpath maintains switching lookup tables that are used to process traffic directly in kernel space. However, flows that are not recognized in the switching tables must be sent to user space for classification so that an appropriate switching rule can be inserted into the fastpath's tables. The new rule is subsequently used in the fastpath to handle previously unrecognized flows. While OvS is largely performant, packets processed by OvS must traverse the kernel network stack and as such are subject to any inherent latency and performance impacts.

The Data Plane Development Kit (DPDK) is a set of libraries and drivers that enable faster packet processing. In 2014 DPDK was integrated with OvS. The result was a set of DPDK-accelerated OvS network devices (netdevs), which enabled packets to be processed solely in user space. The most important advantage of DPDK-enabled netdevs is significant acceleration of I/O traffic between the virtual switch and connected NIC. In some cases, OvS with DPDK performed **12x** faster compared to OvS without DPDK.[1]

DPDK leverages poll mode drivers (PMDs) that continuously scan the NIC for arriving frames. Upon arrival, frames are directly transferred into user-space shared memory using direct memory access. There, OvS processes the frames, bypassing the kernel network stack and thus avoiding the overhead of handling interrupts and context switching, which would otherwise have been experienced in a non-DPDK-enabled setup. Furthermore, the traffic paths between OvS and OpenStack VMs have also been optimized. OvS 2.4 saw the introduction of DPDK-optimized vHost-user netdevs, which improved host-guest communication significantly over VirtIO back ends such as vHost-net.

On the compute nodes of this PoC, we used OvS 2.5.90 with DPDK 16.04 in order to take advantage of multi-queue support. We also contributed to the open source community by enabling TCP segmentation offload (TSO) in OVS-DPDK for flat and VLAN networks. This helped boost packet throughput between VMs.

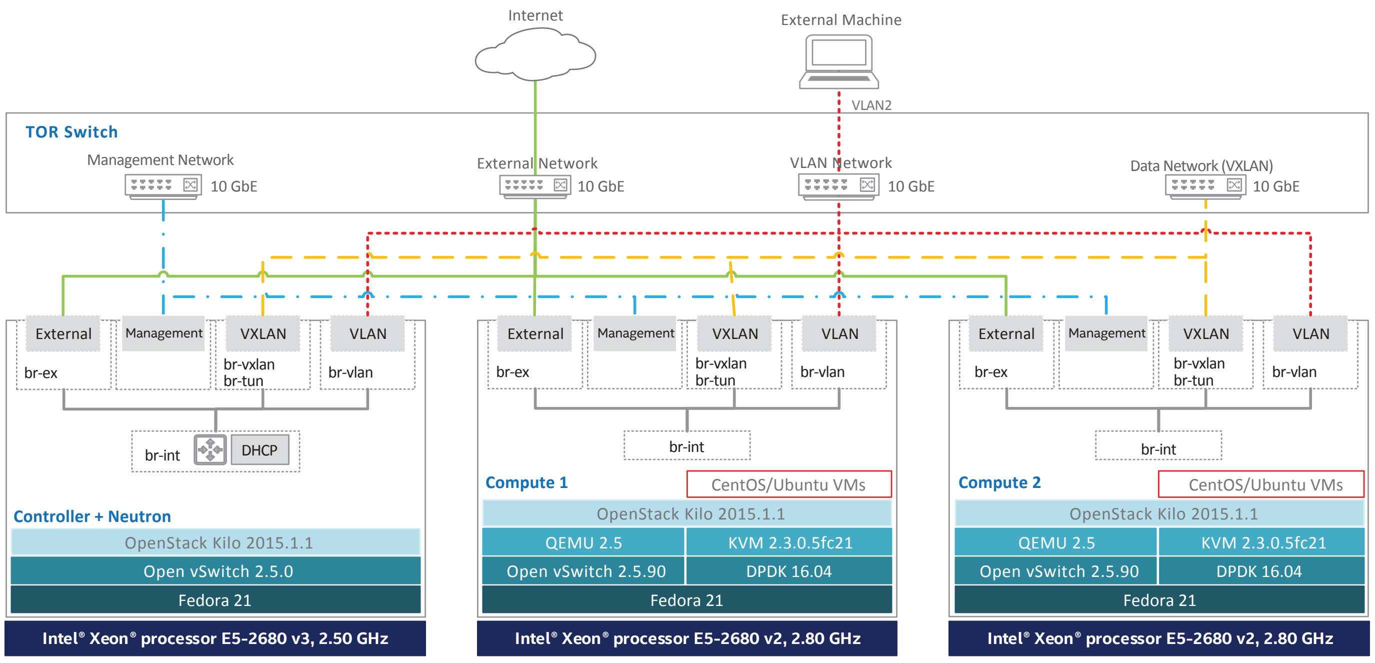

Figure 1. Physical topology of the PoC infrastructure with three Intel® Xeon® processor-based servers in an OpenStack* environment. One server was configured as a controller, and the other two were configured as compute nodes. All servers are connected through a top-of-rack switch and are running Fedora* 21 Server. Each server has four network interfaces, one for each network type as described in Table 1.

Network Topology

In the OpenStack setup, we created two provider networks: an External network and a VLAN network. The External network was a flat (untagged) network, which provided Internet access to OpenStack VMs and was built using Intel® Ethernet Server Adapter I350-T4 (1 GbE). The VLAN network was an 802.1Q tagged network that mapped to the existing physical VLAN network outside the OpenStack cloud. On the compute nodes, Intel® Ethernet Server Adapter X520-DA2NICs were used to connect VMs on this VLAN network to an external machine that was connected to the same physical VLAN network. OVS-DPDK and neutron-ovs-dpdk agents were installed on both compute nodes to enable fast packet processing on these VLAN interfaces.

The Dynamic Host Configuration Protocol (DHCP) service was installed on the controller node and enabled for each OpenStack network. This DHCP service provided OpenStack VMs with IP addresses from a pool of addresses allocated to each subnet. External machines or VMs that were outside the OpenStack cloud were assigned static IP addresses that were outside the pool allocated to the OpenStack subnets.

After OpenStack VMs were spawned on the same VLAN network and separate compute nodes, east-west traffic flowed between the VLAN interfaces of the compute nodes, going through a top-of-rack switch. When VMs were on the same compute node and OpenStack network, switching happened within that compute node and packets never left the host machine. In this PoC, our focus and performance tests were conducted on a single-provider VLAN network. As a result, we did not need a virtual router or tunnel.

Tuning the System for Optimal TCP Traffic Performance

TCP traffic performance tests were started after compiling the plain OvS with DPDK (OVS-DPDK) and setting the *"extra-spec"* property[2] to OpenStack Compute* flavors for OpenStack VMs to make use of features like CPU affinity, huge pages, and single NUMA node topology. Intel® Hyper-Threading Technology (Intel® HT Technology) and Intel® Turbo Boost Technology were enabled in the BIOS of both compute nodes. For highly NUMA-optimized workloads, one can also consider enabling *"Cluster-on-Die"* optimization. In our PoC setup, both compute nodes had processors where the Cluster-on-Die feature was not supported.

Additionally, we set the nova extra-spec property to *"hw:cpu_threads_policy=prefer."* When the host machine has Intel HT Technology enabled, this property guarantees that virtual CPUs (vCPUs) got placed on the same physical core, making them thread siblings (see Table 4). In this technical brief, we refer to all above optimizations as the *"baseline performance tunings."*

As part of our performance testing, OpenStack VMs were spawned with several **variations of virtual resources** such as:

- 1 virtual CPU and 1 GB RAM.
- 2 virtual CPUs and 2 GB RAM, where a single physical core was used to accommodate 2 vCPUs.
- 4 virtual CPUs and 4 GB RAM, where 2 physical cores were used to accommodate 4 vCPUs.
- 8 virtual CPUs and 8 GB RAM, where 4 physical cores were used to accommodate 8 vCPUs.

Performance tests were executed for VMs with two different pairs of Linux* distributions: a pair of CentOS* 7 with kernel 3.10 and a pair of Ubuntu* 14.04 with kernel 3.13. The reason we tested two different Linux distributions was to identify any major differences in performance across two of the most common open source operating systems, especially when tested with their out-of-box kernels. Once both distributions were tested with the baseline performance tunings, the kernel was upgraded to version 4.5.4, which was the latest available kernel at the time of testing.

In all test scenarios and configurations, we used the iPerf3* tool to test TCP traffic throughput. Both iPerf3 server and client VMs were spawned on the OpenStack VLAN network. In this technical brief, a VM with an iPerf3 instance is referred to as *"iPerf3 speed test VM."* All tests were executed for a duration of 60 seconds. In addition, **multiple iPerf3 TCP streams** (1, 2, 4, 8, and 12) were tested in almost all configurations. We also experimented with three **different PMD core masks** to find the optimal assignment of virtual cores to DPDK PMD threads. Table 3 and Figure 8 in the Appendix show the details of how we assigned different cores to different DPDK PMD threads.

Because iPerf3 is not a multi-threaded network bandwidth measurement tool, we ran multiple iPerf3 processes in each iPerf3 speed test VM. Each process was run on a different port and pinned to a different logical core inside the guest VM.

Table 1. Networks used in the PoC.

NETWORK	NETWORK DESCRIPTION	COMPUTE NODES NIC	CONTROLLER NODE NIC
External	Flat provider network used for Internet/remote access to the hosts and OpenStack VMs.	Intel® Ethernet Server Adapter I350-T4	
VLAN	802.1Q tagged provider network mapped to the existing physical virtual local area network (VLAN). This network simulates the subscribers.	Intel® Ethernet Server Adapter X520-DA2	Intel® Ethernet Converged Network Adapter X710-DA4
Management	Management network used for accessing and managing OpenStack* services.	Intel® Ethernet Server Adapter I350-T4	
Data (VxLAN)	Virtual extensible local area network (VxLAN) tunnel used for east-west traffic between tenant VMs on different hosts.	Intel® Ethernet Server Adapter X520-DA2	Intel® Ethernet Converged Network Adapter X710-DA4

Additional Optimizations to Enhance TCP Traffic Performance

We collected the results after implementing all baseline performance tunings and realized that the achieved TCP traffic throughput would be unsatisfactory for high-speed, data-related network functions such as those that CSPs would run. Consequently, we executed the following on our PoC setup:

- **Upgraded the software stack**: We upgraded the software stack on both compute nodes by recompiling OvS 2.5.90 with DPDK 16.04, which also entailed upgrading QEMU* to version 2.5.0.
- **Kernel upgrade**: We upgraded the Linux kernel of all iPerf3 speed test VMs to version 4.5.4. We did not upgrade the host kernel.
- **OvS support for TSO on DPDK vHost-user ports**: TSO support in OVS-DPDK was not available at the time of this testing. Thus, we implemented and published an OVS-DPDK patch to support this. The patch enabled successful feature negotiation of TSO (and implicitly, transmit checksum offloading) between the hypervisor and the OVS-DPDK vHost-user back end. Thus, TSO may be enabled on a per-port basis in the VM using the standard Linux *'ethtool'* utility. Furthermore, the patch also increases the maximum permitted frame length for OVS-DPDK-netdevs to 64 KB—a necessity to accommodate oversized frames received—and provides support for handling *"offload"* frames.

Note that we validated the patch only on OpenStack deployed flat and VLAN networks. The guest may only take advantage of TSO if OvS is connected to a NIC that supports such functionality.

The offloading works as follows: when OvS dequeues a frame from a TSO-enabled guest port using the DPDK vHost library, the library sets specific offload flags in the metadata that DPDK uses to represent a frame (known as *"mbuf"*). Upon receipt of an 'offload' mbuf, OvS sets additional offload flags and attribute values in the mbuf before passing it to the DPDK NIC driver for transmission. The driver examines and interprets the mbuf's offload flags and corresponding attributes to facilitate TCP segmentation on the NIC.

With the enablement of TSO for DPDK netdevs in OvS, the segmentation of guest-originated oversized TCP frames moves from the guest operating system's software TCP/IP stack to the NIC hardware. The benefits of this approach are many. First, offloading segmentation of a guest's TCP frames to hardware significantly reduces the compute burden on the VM's vCPU. Consequently, when the guest does not need to segment frames itself, its vCPU can take advantage of the additionally available computational cycles to perform more meaningful work.

Second, with TSO enabled, OvS does not need to receive, process, and transmit a large number of smaller frame segments, but rather a smaller amount of significantly larger frames. In other words, the same amount of data can be handled with significantly reduced overhead.

Finally, the reduction in the number of small packets, which are sent to the NIC for transmission, results in the reduction of Peripheral Component Interconnect* bandwidth usage. The cumulative effect of these enhancements is a massive improvement in TCP throughput for DPDK-accelerated OvS.

Our TSO patch for OvS 2.5.90 and DPDK 16.04 was published as a request for comment on the ovs-dev mailing list and is available for you to download from https://mail.openvswitch.org/pipermail/ovs-dev/2016-June/316414.html.

- **Multi-queue support for DPDK vHost-user ports**: In this upgraded stack, multi-queue was available for us to use. Multi-queue enables VMs to scale network performance as the number of virtual CPUs increases.

Test Scenarios

We set up several test scenarios and did performance testing with various configurations and optimizations. In this technical brief, we focus on only three test scenarios that are the closest to a real-world setup.

Test Scenario #1: Multiple external iPerf3 clients connected to an iPerf3 server VM residing inside an OpenStack-managed cloud

On an external server outside the OpenStack cloud, we spawned five CentOS 7 VMs on the VLAN2 network. These VMs simulate subscriber requests that connect to a broadband speed test server, which reside inside the OpenStack-managed cloud. VMs are running on top of the Kernel-based Virtual Machine (KVM*) and OvS (see Figure 2).

There were no optimizations done to the external client VMs; however, each VM had allocated to it two virtual CPUs and 2 GB of RAM. The iPerf3 speed test server VM was spawned inside the OpenStack-managed cloud and was initially optimized with the baseline performance tunings mentioned above. After recording initial test results, we applied all additional TCP optimizations mentioned above to the iPerf3 speed test VM (except for multi-queue) and recaptured the results.

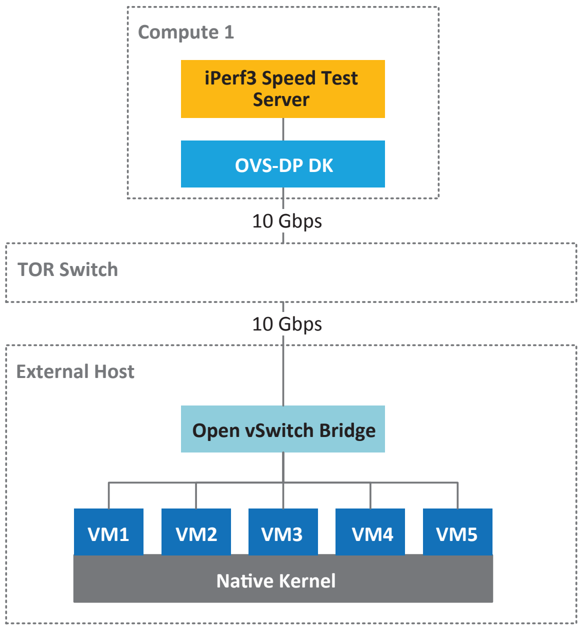

Figure 2. Test Scenario 1—Multiple external clients connected to iPerf3* speed test server.

Test Scenario #2: iPerf3 client VM connected to an iPerf3 server VM, with both VMs located on different compute nodes within an OpenStack-managed cloud

Two CentOS 7 VMs were spawned on different compute nodes and on the same OpenStack VLAN2 network. One VM was configured with an iPerf3 client to simulate any network function interacting with TCP traffic (for example, responding to TCP data or forwarding TCP data), while the other VM was configured with an iPerf3 server that simulated a broadband speed test server (see Figure 3).

Both VMs and both compute nodes were initially optimized with baseline performance tunings, and the results were captured. Next, we applied all additional TCP optimizations mentioned above to the iPerf3 speed test VM (except for multi-queue) and recaptured the results. Finally, we repeated the scenario with multi-queue enabled and, recaptured again the results.

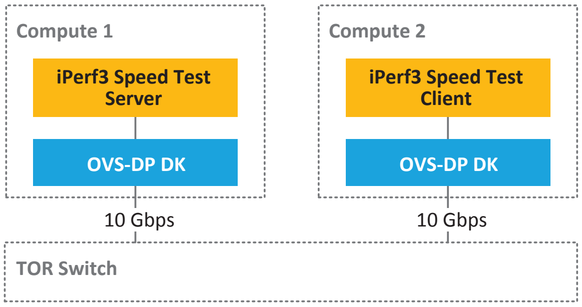

Figure 3. Test Scenario 2—iPerf3* client and iPerf3 server VMs are spawned on different compute nodes.

Test Scenario #3: iPerf3 client VM connected to an iPerf3 server VM, with both VMs located on the same compute node within an OpenStack-managed cloud

Two CentOS 7 VMs were spawned on the same compute node and connected to the OpenStack VLAN2 network. One VM was configured as the iPerf3 client, while the other VM was configured as an iPerf3 server (see Figure 4).

Both VMs and the compute node were initially optimized with baseline performance tunings, and the results were captured. Next, we applied all additional TCP optimizations mentioned above to the iPerf3 speed test VM (except for multi-queue), and recaptured results. Finally, we repeated the scenario with multi-queue enabled, and recaptured again the results.

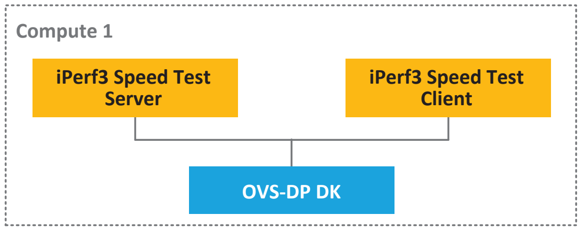

Figure 4. Test Scenario 3—iPerf3* client and iPerf3 server are spawned on the same compute node.

Results

When VMs were running operating systems with default kernels, we noticed that TCP throughput results were generally better on Ubuntu 14.04 VMs than on CentOS 7 VMs. However, after upgrading the kernels of both Ubuntu and CentOS VMs to version 4.5.4, we experienced comparable performance results. For this reason, we are presenting only the CentOS 7 results.

For Test Scenario #1, we were able to achieve an accumulative average throughput of **3.87 Gbps** using baseline performance tunings (see the left bar of Figure 5). After upgrading the Linux kernel to version 4.5.4 in the speed test server VM, upgrading the software stack and enabling TSO, the accumulative average TCP traffic throughput increased to **9.35 Gbps** (see the right bar of Figure 5). This is around a **2.4x** improvement. Both results were obtained when we used two logical cores from separate physical cores for DPDK PMD threads, and when two vCPUs and 2 GB RAM were allocated to the iPerf3 server VM.

For Test Scenario #2, using the baseline TCP tunings we achieved an average throughput between **4.6 Gbps** and **5.5 Gbps**, depending on the number of iPerf3 streams used (see Figure 6). After applying the additional TCP tunings (with and without multi-queue), we the average throughput increased to around **9.35 Gbps**, which was an improvement of approximately **1.7x**. This shows that multi-queue had a negligible effect in this scenario. Notice that these results were obtained when one logical core was assigned to the DPDK PMD thread, and client and server VMs were allocated two vCPUs and 2 GB RAM each.

For Test Scenario #3, using the baseline TCP tuning we achieved an average throughput of **3.9 Gbps** (see Figure 7). After applying the additional TCP tunings without multi-queue, the throughput ranged between 22 and 33.5 Gbps, depending on the number of iPerf3 streams. With additional eight queues enabled, we were able to achieve a throughput of around **45 Gbps** for larger number of streams. The impact of applying all TCP tunings in this scenario resulted in an approximately **10x** higher average TCP throughput. Notice that these results were obtained when we used two logical cores from separate physical cores for DPDK PMD threads, and when four vCPUs and 4 GB RAM were allocated to the iPerf3 speed test VMs.

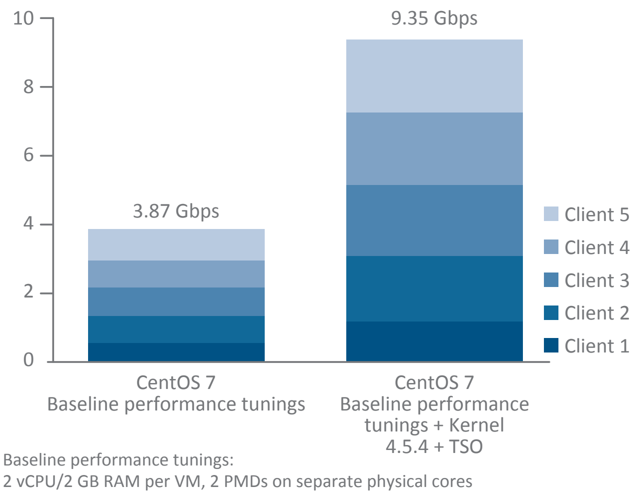

Figure 5. Test Scenario #1—an increase of ~2.4x in TCP throughput.

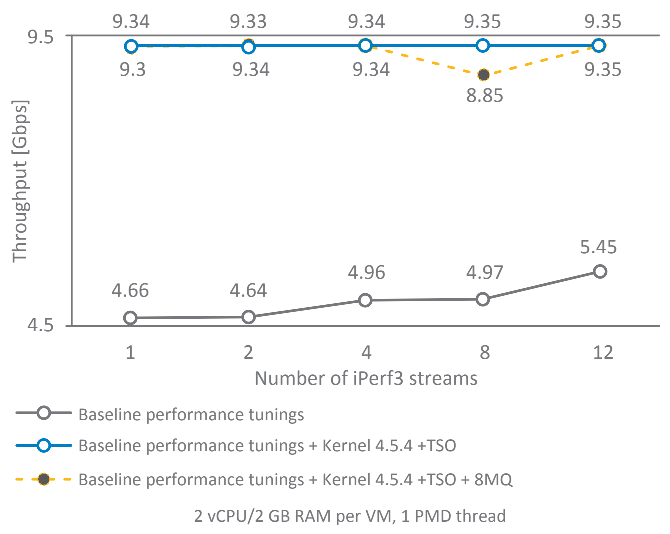

Figure 6. Test Scenario #2—an increase of ~1.7x in TCP throughput.

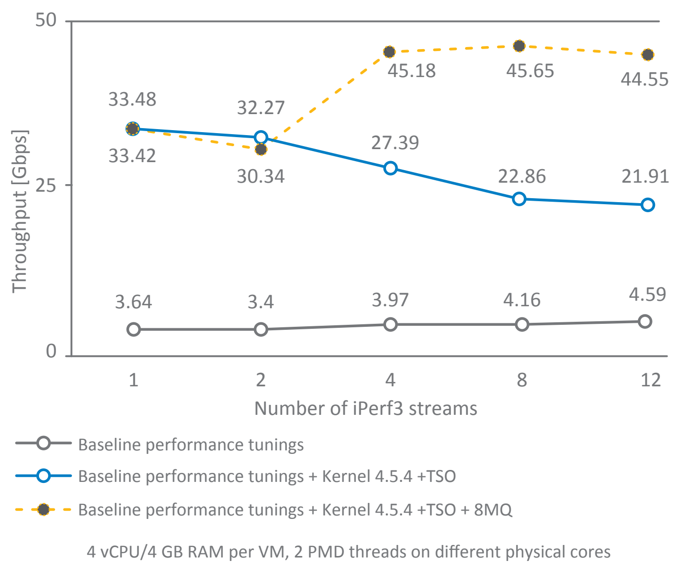

Figure 7. Test Scenario #3—an increase of ~10x in TCP throughput.

Summary

The most impressive TCP performance boost was achieved after enabling TSO for DPDK netdevs in OvS and utilizing the multi-queue feature in speed test VMs. At the time of this writing, multi-queue support for OVS-DPDK was made available in the OvS mainline, while work on upstreaming of TSO support was in progress. For example, patches for supporting TSO for flat networks and VLAN networks in OVS-DDPK are currently available here: https://mail.openvswitch.org/pipermail/ovs-dev/2016-June/316414.html.

As a result of enabling TSO and upgrading the Linux kernel to 4.5.4, we achieved up to a **1.7x** higher throughput when client and server speed test VMs were spawned on different compute nodes, compared to a system where only baseline performance tunings were applied. When both speed test VMs were deployed on the same compute node, enabling the multi-queue feature allowed packet processing to improve further by approximately **10x**.

Open source software provides a multitude of ways for optimizing packet processing performance on COTS servers. With the use of Intel Xeon processors and a high-speed network infrastructure, including 10 Gigabit Intel® Ethernet Controllers, virtualization of more high-speed network function workloads is feasible. “Intel Architecture delivers a high-performance PoC infrastructure capable of handling heavy workloads.

Some open source projects are already mature enough to offer a gamut of enhancements dedicated to cloud and virtualized environments. OpenStack Kilo provides provisions to optimize performance for its VMs. Some of the key features that improve overall performance include NUMA topology awareness and huge pages, which reduce the number of translation lookaside buffer (TLB) misses. Other features include affinitizing a VM’s CPU cores to the host machine. Proper selection from a variety of Linux distributions may be relevant, as they come with different kernel versions, and may perform differently over the same infrastructure. Our tests showed that performance of packet processing improved with the latest release of the kernel. However, due to the large amount of TCP-related updates and fixes integrated with each kernel version, it is difficult to pinpoint exactly which update impacted the overall performance.

The next step on the optimization path is the selection of packet processing acceleration technologies. We strongly recommend the use of DPDK because it provides a flexible and fast packet-forwarding path between VMs and the physical NIC of the hosts on which they reside. In this PoC, we used OVS-DPDK to benefit from fast packet switching and forwarding. Since DPDK PMDs continuously scan the host’s NICs for arriving data, they require dedicated hardware resources. In most of our testing, we achieved optimal performance by dedicating two non-sibling logical cores to the DPDK PMD threads.

Today’s open source projects are evolving rapidly with new optimizations being developed all the time. Be sure to research these enhancements thoroughly and use the right optimizations that are relevant to your workload and test scenarios. The optimizations we presented in this technical brief applied best to an iPerf3 TCP workload, but your workloads may have different requirements. Consequently, not all optimizations mentioned in this technical brief might be applicable.

Appendix

Hardware Details

Table 2. Dual-processor COTS servers based on Intel® Xeon® processors provide a multitude of processing cores and high speed networking to serve as powerful and energy-efficient platforms for virtualized workloads.

	CONTROLLER / NEUTRON	COMPUTE 1	COMPUTE 2
Processor	2x Intel® Xeon® processor E5-2680 v3, 2.50 GHz, 48 logical cores with Intel® Hyper-Threading Technology (Intel® HT Technology)	2x Intel® Xeon® processor E5-2680 v2, 2.80 GHz, 40 logical cores with Intel® HT Technology	2x Intel® Xeon® processor E5-2680 v2, 2.80 GHz, 40 logical cores with Intel® HT Technology
Memory	128 GB, DDR4-2133	64 GB, DDR3-1600	64 GB, DDR3-1600
Management and External networks NICs	Intel® Ethernet Server Adapter I350-T4	Intel® Ethernet Server Adapter I350-T4	Intel® Ethernet Server Adapter I350-T4
Data (VXLAN) and VLAN networks NICs	Intel® Ethernet Converged Network Adapter X710-DA4	Intel® Ethernet Server Adapter X520-DA2	Intel® Ethernet Server Adapter X520-DA2 1 TB HDD
Storage	200 GB HDD	1 TB HDD	
Top-of-rack switch	Extreme Networks Summit* X670V-48t-BF-AC 10GbE Switch, SFP+ Connections		

Details on Performance Tunings

1. **OVS-DPDK** and **neutron-ovs-dpdk** agents were installed on both compute nodes.
2. **1 GB huge pages** were used for OpenStack VMs to reduce TLB misses by memory management hardware and CPU on x86_64 architectures.
3. **CPU pinning schema:** Each compute has 20 physical cores spread across two NUMA nodes, 10 in each NUMA node. We enabled Intel HT Technology in those systems to get a total of 40 logical cores per compute node and 20 logical cores in each NUMA node:
   - On both compute nodes, 1 physical core and its sibling core were dedicated for Linux scheduling and host housekeeping.
   - DPDK PMD threads were pinned to CPU cores on the host machine. We tested three different pmd-core-masks configurations as presented in Figure 8. In most of the over 2,000 tests results, the best throughput was achieved when two physical (non-sibling) cores were assigned to DPDK PMD threads and both CPU cores belonged to the same NUMA node as the NIC. Second best results were seen when one logical core was assigned/dedicated for DPDK PMD threads. Throughput results were not as good when we used two logical cores, from the same physical core, for test scenario 1 and 3. Please note that these results are optimal for the presented workload and test scenarios. You might need to consider using a different pmd-core-mask for your workloads.
   - Remaining cores of the NUMA node were allocated to OpenStack compute services as shown in Table 3

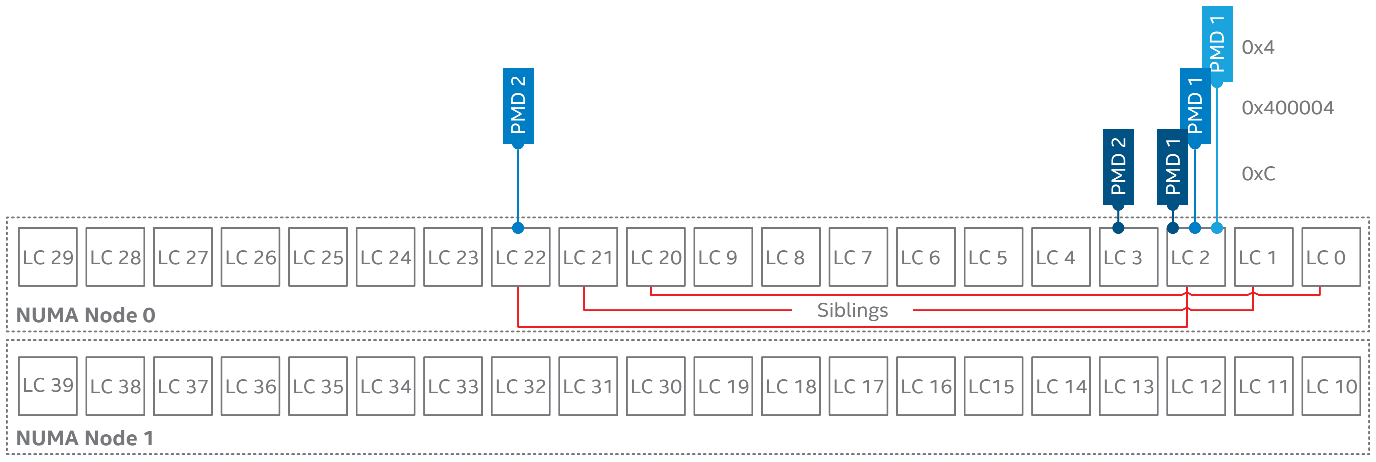

Figure 8. Logical core numbering on both compute nodes used in the PoC. Only NUMA node 0 is used. There is an offset of 20 between each of the sibling cores. 0x400004 refers to pmd-core-mask when two logical cores (LC2 and LC22) on the same physical core are dedicated for the DPDK PMD threads. 0x4 refers to pmd-core-mask, when one logical core (LC2) was dedicated for a PMD thread, and 0xC refers to pmd-core-mask when two logical cores (LC2 and LC3) on different physical cores are dedicated for the PMD threads.

Table 3. CPU core pinning schema. Cores 11–19 and 31–39, from NUMA node 1, are unassigned and not used in performance testing.

NUMA 0 NODE CORES	DESCRIPTION	CONFIGURING
0, 20	Housekeeping	In /etc/default/grub file: GRUB_CMDLINE_LINUX = ... isolcpus=1–19,21–39
2, 3	Open vSwitch*-Data Plane Development Kit poll mode driver threads	In /etc/default/ovs-dpdk file: OVS_PMD_CORE_MASK=C Or using command: ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=C
1	ovs-vswitchd daemon	In /etc/default/ovs-dpdk file: OVS_CORE_MASK=2
4–9, 24–29	OpenStack* virtual machines	In /etc/nova/nova.conf file: vcpu_pin_set = 4–9,11–19,24–29,31–39

4. **Optimizations for OpenStack VMs**: We used special OpenStack flavors to spawn iPerf3 speed test VMs. We set extra_specs properties to those flavors, as mentioned in Table 4, to make sure that:

- Network interface card, DPDK PMD threads, and both the virtual CPU and memory of the VMs fall on the same NUMA node 0 of the compute node.
- OpenStack VMs use dedicated CPUs, and guest's CPUs are pinned to host's CPUs.
- OpenStack VMs use 1 GB huge pages.
- Guest vCPUs are placed on same physical core whenever possible.

Table 4. The *'extra-specs'* settings for iPerf3* speed test VMs.

EXTRA_SPECS PARAMETER	VALUE	NOTES
hw:cpu_policy	dedicated	Guest virtual CPUs will be strictly pinned to a set of host physical CPUs.
hw:mem_page_size	large	Guest will use 1 GB huge pages.
hw:numa_mempolicy	preferred	Provide memory resources according to extra_specs but if more resources are needed take from other NUMA node.
hw:numa_mem.0	4096	Mapping MB of RAM to NUMA node 0.
hw:numa_nodes	1	Number of NUMA nodes to expose to the guest.
hw_numa_cpus.0	0,1,2,3	Mapping of virtual CPUs list to NUMA node 0.
hw:cpu_threads_policy	prefer	If the host has threads, vCPU will be placed on the same core, so they are thread siblings.

Table 5. iPerf3* execution commands.

SCENARIO	iPerf3 SERVER VM EXECUTION	iPerf3 CLIENT VM EXECUTION
Scenario 1	iperf3 –s –p2000 -D iperf3 –s –p3000 -D iperf3 –s –p4000 -D iperf3 –s –p5000 -D iperf3 –s –p6000 -D	iPerf3 client VM #1: iperf3 –c <*SERVER_IP*> -t60 -p2000 & iPerf3 client VM #2: iperf3 –c <*SERVER_IP*> -t60 –p3000 & iPerf3 client VM #3: iperf3 –c <*SERVER_IP*> -t60 –p4000 & iPerf3 client VM #4: iperf3 –c <*SERVER_IP*> -t60 –p5000 & iPerf3 client VM #5: iperf3 –c <*SERVER_IP*> -t60 –p6000 &
Scenario 2 (2 iPerf3 processes on 2 virtual CPUs)	iperf3 –s –A0 –p2000 -D iperf3 –s –A1 –p3000 -D	iperf3 –c <*SERVER_IP*> -A0,0 -t60 –P <*NO_OF_STREAMS*> -p2000 & iperf3 –c <*SERVER_IP*> -A1,1 -t60 –P <*NO_OF_STREAMS*> -p3000 &
Scenario 3 (4 iPerf3 processes on 4 virtual CPUs)	iperf3 –s –A0 –p2000 -D iperf3 –s –A1 –p3000 -D iperf3 –s –A2 –p4000 -D iperf3 –s –A3 –p5000 -D	iperf3 –c <*SERVER_IP*> -A0,0 -t60 –P <*NO_OF_STREAMS*> -p2000 & iperf3 –c <*SERVER_IP*> -A1,1 -t60 –P <*NO_OF_STREAMS*> -p3000 & iperf3 –c <*SERVER_IP*> -A2,2 -t60 –P <*NO_OF_STREAMS*> -p4000 & iperf3 –c <*SERVER_IP*> -A3,3 -t60 –P <*NO_OF_STREAMS*> -p5000 &

Acronyms

NAME	REFERENCE
COTS	Commercial Off-the-Shelf
CPU	Central Processing Unit
CSP	Communication Service Provider
DHCP	Dynamic Host Configuration Protocol
DOCSIS	Data Over Cable Service Interface Specification
DPDK	Data Plane Development Kit
I/O	Input/Output
Intel® HT Technology	Intel® Hyper-Threading Technology
KVM	Kernel-based Virtual Machine
LC	Logical Core
netdev	Network Device
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NIC	Network Interface Card
NUMA	Non-Uniform Memory Architecture
OvS	Open vSwitch

NAME	REFERENCE
OVS-DPDK	DPDK-Accelerated Open vSwitch
PMD	Passive Optical Network
PoC	Quick Emulator
PON	Random Access Memory
QEMU	Software Defined Networking
RAM	Transmission Control Protocol
SDN	Transaction Lookaside Buffer
TCP	TCP Segmentation Offload
TLB	User Datagram Protocol
TSO	Virtual CPU
UDP	Virtual Local Area Network
vCPU	Virtual Machine
VLAN	Virtual Network Function
VM	Virtual Switch
VNF	Virtual eXtensible Local Area Network
vSwitch	Virtual Tunnel End Point
VxLAN	Virtual eXtensible LAN

[1] https://networkbuilders.intel.com/docs/open-vswitch-enables-sdn-and-nfv-transformation-paper.pdf

[2] These are extra specifications that OpenStack administrators can set. See http://docs.openstack.org/admin-guide/compute-flavors.html