Intel® Open Network Platform Server Release 1.5 Performance Test Report

SDN/NFV Solutions with Intel® Open Network Platform Server

Document Revision 1.1
October 2015

Revision History

Date	Revision	Comments
October 2nd, 2015	1.1	Fixed some broken links
September 30, 2015	1.0	Initial document for release of Intel® Open Network Platform Server Release 1.5

Contents

Figures

Tables

1.0 Audience and Purpose

Intel® Open Network Platform Server (Intel ONP Server) is a Reference Architecture that provides engineering guidance and ecosystem-enablement support to encourage widespread adoption of Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) solutions in Telco, Enterprise, and Cloud. Intel® Open Network Platform Server Reference Architecture Guides and Release Notes are available on 01.ORG.

The primary audiences for this test report are architects and engineers implementing the Intel® Open Network Platform Server Reference Architecture using open-source software ingredients that include:

- DevStack*
- OpenStack*
- OpenDaylight*
- Data Plane Development Kit (DPDK)*
- Open vSwitch (OVS)*
- Fedora*

This test report provides a guide to packet processing performance testing of the Intel® ONP Server. The report includes baseline performance data and provides system configuration and test cases relevant to SDN/NFV. The purpose of documenting these configurations and methods is not to imply a single "correct" approach, but rather to provide a baseline of well-tested configurations and test procedures. This will help guide architects and engineers who are evaluating and implementing SDN/NFV solutions more generally and can greatly assist in achieving optimal system performance.

2.0 Summary

Benchmarking an SDN/NFV system is not trivial and requires expert knowledge of networking and virtualization technologies. Engineers also need benchmarking and debugging skills, as well as a good understanding of the device-under-test (DUT) across compute, networking, and storage domains. Knowledge of specific network protocols and hands-on experience with relevant open-source software such as Linux, kernel-based virtual machine (KVM), quick emulator (QEMU), DPDK, OVS, etc., are also required.

Repeatability is very important when testing complex systems and can be difficult to achieve with manual methods. Scripted install procedures and automated test methods will be needed for developing SDN/NFV solutions. Future versions of Intel® ONP Server will address this critical aspect.

This report builds on earlier Intel® ONP Server test reports available on 01.ORG as archived documents. A previous report (Intel ONP Server 1.3) contains certain baseline throughput test data and procedures for Linux operating system setup, BIOS configurations, core-usage configuration for OVS, VM setup, and building DPDK and OVS.

This current test report includes the following:

- New versions of software ingredients
- vHost user for QEMU
- 40Gbps performance testing with the Intel® Ethernet X710-DA2 Adapter
- Latency performance metrics
- Tuning methods and troubleshooting tips for OVS
- Information on related industry NFV test activities

Performance data include the following configurations:

- Host Tests
- Virtual Switching Tests
- PHY-to-VM Tests
- VM-to-VM Tests

3.0 Platform Specifications

3.1 Hardware Ingredients

Table 3–1. Hardware Ingredients

Item	Description
Server Platform	Intel® Server Board S2600WT2 DP (code-named Wildcat Pass) Two processors per platform
Chipset	Intel® C610 series chipset (code-named Wellsburg)
Processor	Intel® Xeon® Processor E5-2697 v3 (code-named Haswell) Speed and power: 2.60 GHz, 145 W Cache: 35 MB per processor Cores: 14 cores, 28 hyper-threaded cores per processor for 56 total hyper-threaded cores QPI: 9.6 GT/s Memory types: DDR4-1600/1866/2133, Reference: http://ark.intel.com/products/81059/Intel-Xeon-Processor-E5-2697-v3-35M-Cache-2_60-GHz
Memory	Micron 16 GB 1Rx4 PC4-2133MHz, 16 GB per channel, 8 Channels, 128 GB Total
Local Storage	500 GB HDD Seagate SATA Barracuda 7200.12 (SN:9VMKQZMT)
PCIe	Port 3a and Port 3c x8
NICs	2 x Intel® Ethernet CAN X710-DA2 Adapter (Total: 4 x 10GE Ports) (code-named Fortville)
BIOS	Version: SE5C610.86B.01.01.0008.021120151325 Date: 02/11/2015

3.2 Software Version

Table 3–2. Software Versions

System Capability	Version
Host Operating System	Fedora 21 x86_64 (Server version) Kernel version: 3.17.4-301.fc21.x86_64
VM Operating System	Fedora 21 (Server version) Kernel version: 3.17.4-301.fc21.x86_64
Libvirt	libvirt-1.2.9.3-2.fc21.x86_64
QEMU	QEMU-KVM version 2.2.1 http://wiki.qemu-project.org/download/qemu-2.2.1.tar.bz2
DPDK	DPDK 2.0.0 http://www.dpdk.org/browse/dpdk/snapshot/dpdk-2.0.0.tar.gz
OVS	Open vSwitch 2.4.0 http://openvswitch.org/releases/openvswitch-2.4.0.tar.gz

3.3 Boot Settings

Table 3–3. Boot Settings

System Capability	Description
Host Boot Settings	HugePage size = 1 G ; no. of HugePages = 16 HugePage size = 2 MB; no. of HugePages = 2048 intel_iommu=off Hyper-threading disabled: isolcpus = 1-13,15-27 Hyper-threading enabled: isolcpus = 1-13,15-27,29-41,43-55
VM Kernel Boot Parameters	GRUB_CMDLINE_LINUX="rd.lvm.lv=fedora-server/root rd.lvm.lv=fedora-server/swap default_hugepagesz=1G hugepagesz=1G hugepages=1 hugepagesz=2M hugepages=1024 isolcpus=1,2 rhgb quiet"

3.4 Compile Options

Table 3–4. Compile Option Configurations

System Capability	Configuration
DPDK Compilation	CONFIG_RTE_BUILD_COMBINE_LIBS=y CONFIG_RTE_LIBRTE_VHOST=y CONFIG_RTE_LIBRTE_VHOST_USER=y DPDK compiled with "-Ofast -g"

OVS Compilation	OVS configured and compiled as follows:
	`./configure --with-dpdk=<DPDK SDK PATH>/x86_64-native-linuxapp \ CFLAGS="-Ofast -g"`
	`make CFLAGS="-Ofast -g -march=native"`

DPDK Forwarding Applications	Build L3fwd: (in l3fwd/main.c)
	`#define RTE_TEST_RX_DESC_DEFAULT 2048`
	`#define RTE_TEST_TX_DESC_DEFAULT 2048`
	Build L2fwd: (in l2fwd/main.c)
	`#define NB_MBUF 16384`
	`#define RTE_TEST_RX_DESC_DEFAULT 2048`
	`#define RTE_TEST_TX_DESC_DEFAULT 2048`
	Build testpmd: (in test-pmd/testpmd.c)
	`#define RTE_TEST_RX_DESC_DEFAULT 2048`
	`#define RTE_TEST_TX_DESC_DEFAULT 2048`

3.5 Operating System Settings

Table 3–5. Software Versions

System Capability	Settings
Linux OS Services Settings	# systemctl disable NetworkManager.service # chkconfig network on # systemctl restart network.service # systemctl stop NetworkManager.service # systemctl stop firewalld.service # systemctl disable firewalld.service # systemctl stop irqbalance.service # killall irqbalance # systemctl disable irqbalance.service # service iptables stop # echo 0 > /proc/sys/kernel/randomize_va_space # SELinux disabled # net.ipv4.ip_forward=0
Uncore Frequency Settings	Set uncore frequency to the max ratio: setting uncore frequency: 1. # rdmsr -p 0 0x620 2. # c1e 3. # rdmsr -p 14 0x620 4. # c1e 5. # wrmsr -p 0 0x620 0x1e1e 6. # wrmsr -p 14 0x620 0x1e1e
PCI Settings	setpci –s 00:03.0 184.l 0000000 setpci –s 00:03.2 184.l 0000000 setpci –s 00:03.0 184.l=0x1408 setpci –s 00:03.2 184.l=0x1408
Linux Module Settings	rmmod ipmi_msghandler rmmod ipmi_si rmmod ipmi_devintf

4.0 Test Configurations

The test setup is shown in Figure 4–1. The system-under-test is Intel® Open Network Platform Server Reference Architecture (version 1.5). The traffic is generated by Ixia running RFC2544 (IxNetwork 7.40.929.15 EA; Protocols: 4.40.1075.13; IxOS: IxOS 6.80.1100.7 EA). The maximum theoretical system forwarding throughput is 40Gbps aggregated across four 10GE ports. Physical ports are paired (one ingress and one egress), i.e., one 10Gbps bidirectional flow "consumes" two ports. Unless otherwise stated, all tests are for zero packet loss.

The VM network interface used is vhost-user with DPDK acceleration. Vhost-user information is available at http://dpdk.readthedocs.org/en/latest/prog_guide/vhost_lib.html along with DPDK 2.0 documentation.

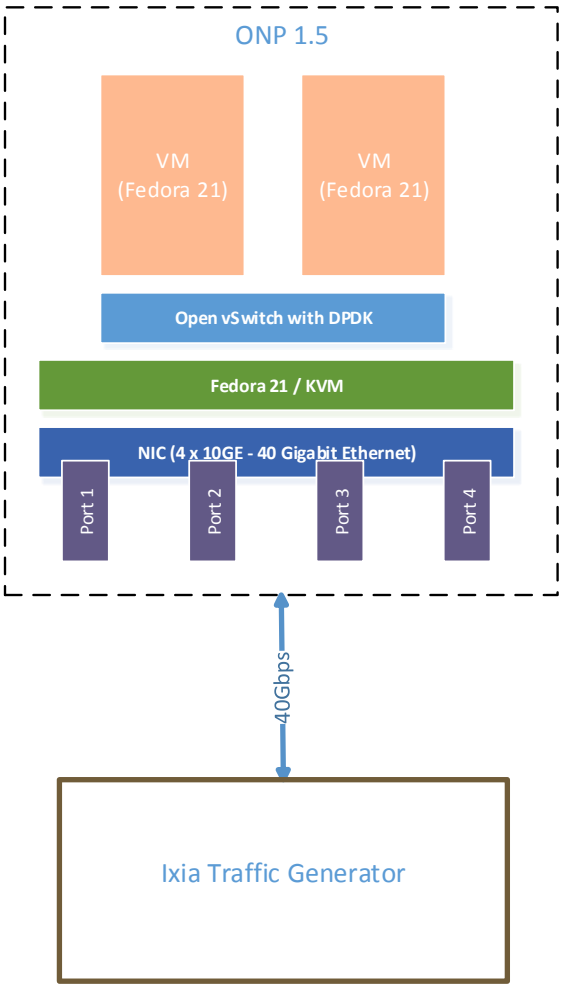


Figure 4–1. High-Level Overview of Test Setup

Allocation of cores has a large impact on performance. For tests in this document, core configurations include physical and hyper-threaded options. Tests showing the impact on performance when adding cores and using hyper-threading are included. Table 4–1 shows combinations of physical and hyper-threaded cores used for various test cases.

Table 4–1. Number of Cores Used for Each Test Category

Test	Physical Cores				Hyper-Threaded Cores			
	1	2	3	4	1	2	3	4
Host Tests				✓				
Virtual Switching Tests	✓	✓		✓		✓		✓
PHY-to-VM Tests	✓							
VM-to-VM Tests	✓							

4.1 Traffic Profiles

The IP traffic profile used conforms to RFC2544 (https://www.ietf.org/rfc/rfc2544.txt).

- Frame sizes (bytes): 64, 128, 256, 512, 1024, 1280, 1518
- Protocol: IPv4
- All tests are bidirectional with the same data rate being offered from each direction.
- Test duration (per packet size): 60 seconds, except for latency and Packet Delay Variation (PDV) soak tests, which are run for 1 hour.

5.0 Test Metrics

5.1 Packet Processing Performance Metrics

RFC 2544 is an Internet Engineering Task Force (IETF) RFC that outlines a benchmarking methodology for network interconnect devices. The methodology results in performance metrics (e.g., latency, frame loss percentage, and maximum data throughput).

In this document, network "throughput" (measured in millions of frames per second) is based on RFC 2544, unless otherwise noted. "Frame size" refers to Ethernet frames ranging from the smallest frames of 64 bytes to the largest of 1518 bytes.

RFC 2544 types of tests are as follows:

- **Throughput tests** define the maximum number of frames per second that can be transmitted without any error. Throughput is the fastest rate at which the count of test frames transmitted by the DUT is equal to the number of test frames sent to it by the test equipment. Test time during which frames are transmitted must be at least 60 seconds.
- **Latency tests** measure the time required for a frame to travel from the originating device through the network to the destination device.
- **Frame loss tests** measure the network's response in overload conditions—a critical indicator of the network's ability to support real-time applications in which a large amount of frame loss rapidly degrades service quality.
- **Burst tests** assess the buffering capability of a switch. They measure the maximum number of frames received at full-line rate before a frame is lost. In carrier Ethernet networks, this measurement validates the excess information rate as defined in many SLAs.
- **System recovery tests** characterize speed of recovery from an overload condition.
- **Reset tests** characterize the speed of recovery from device or software reset.

"Test duration" refers to the measurement period for, and particular packet size with, an offered load and assumes the system has reached a steady state. Using the RFC2544 test methodology, this is specified as at least 60 seconds.

5.2 Throughput

The throughput test data provided in this document represents "platform throughput" as measured by the Ixia traffic generator. Switching performance metrics include the number of switching operations for the particular configuration. This is shown in Table 5–1 using two examples of configurations with two and three switching operations, respectively. Figure 5–1 shows the two configuration examples.

Table 5–1. Throughput and Switching Performance Metrics

Parameter	Configuration Examples	
	PHY-OVS-PHY (four ports)	PHY-VM-VM-PHY (two ports)
Physical Ports	4	2
Flows per Port (in each direction)	1	1
Total Flows	4	2
Switching Operations	2	3
Throughput (packets/sec) 128B packets	33,783,781 100% of line rate	2,830,877 16.8% of line rate
Switching Performance (packets/sec) 128B packets	67,567,562 200% of line rate	5,661,754 50.4% of line rate

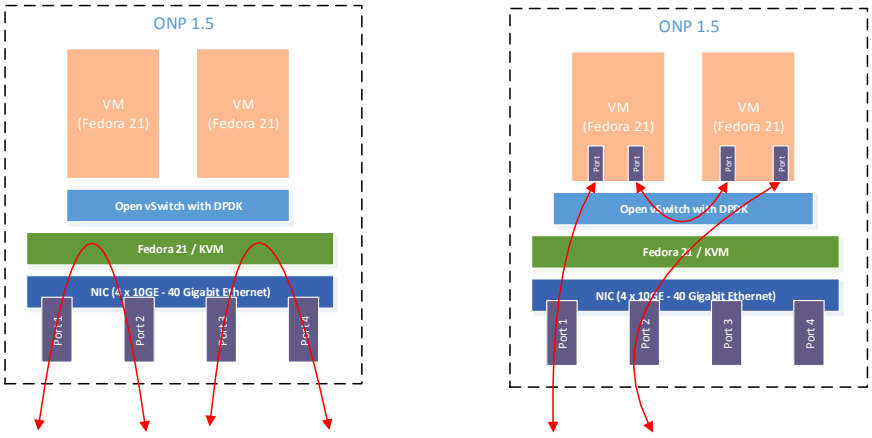


Figure 5–1. Examples of Configurations with Two and Three Switching Operations

5.2.1 Layer 2 Throughput

This test determines the DUT's maximum Layer 2 forwarding rate without traffic loss, as well as average and minimum/maximum latency for different packet sizes.

This test is performed full duplex with traffic transmitting in both directions.

The DUT must perform packet parsing and Layer 2 address lookups on the ingress port, and then modify the header before forwarding the packet on the egress port.

5.2.2 Layer 3 Throughput

This test determines the DUT's maximum IPv4 Layer 3 forwarding rate without packet loss, as well as average and minimum/maximum latency for different packet sizes.

This test is performed full duplex with traffic transmitting in both directions.

The DUT must perform packet parsing and route lookups for Layer 3 packets on the ingress port and then forward the packet on the egress port without modifying the header.

5.3 Latency

With latency (i.e., packet delay) and packet delay variation, it is generally the worst-case performance that must be considered. Outliers can create customer disappointment at the carrier scale and cost service providers.

The RFC2544 measurement of latency is extensively used in traditional testing. NFV requires more information on latency, including packet delay variation. Ideally, the delay of all packets should be considered, but in practice some form of sampling is needed (this may not be periodic sampling).

Average and minimum/maximum latency numbers are usually collected with throughput tests; however, the distribution of latency is a more meaningful metric (i.e., a test that collects latency distribution for different packet sizes and over an extended duration to uncover outliers; latency tests should run for at least 1 hour and ideally for 24 hours). Collecting test data for all traffic conditions can take a long time. One approach is to use the highest throughput that has demonstrated zero packet loss for each packet size as determined with throughput tests.

RFC 2679 defines a metric for one-way delay of packets across Internet paths and describes a methodology for measuring "Type-P-One-way-Delay" from source to destination.

5.4 Packet Delay Variation (PDV)

RFC 3393 provides definitions of PDV metrics for IP packets and is based on RFC 2679. This RFC notes that variation in packet delay is sometimes called "jitter" and that this term causes confusion because it is used in different ways by different groups of people. The ITU-T also recommends various delay variation metrics [Y.1540] [G.1020]. Most of these standards specify multiple ways to quantify PDV.

RFC 5481 specifies two forms of packet delay variation:

- Inter-Packet Delay Variation (IPDV) is where the reference is the previous packet in the stream (according to a sending sequence), and the reference changes for each packet in the stream. In this formulation, properties of variation are coupled with packet sequence. This form was called

Instantaneous Packet Delay Variation in early IETF contributions and is similar to the packet spacing difference metric used for inter-arrival jitter calculations in RFC3550.

- Packet Delay Variation (PDV) is where a single reference is chosen from the stream based on specific criteria. The most common criterion for the reference is the packet with the minimum delay in the sample. This term derives its name from a similar definition for Cell Delay Variation, an ATM performance metric [I.356].

Both metrics are derived from "one-way-delay" metrics and, therefore, require knowledge of time at the source and destination. Results are typically represented by histograms showing statistical distribution of delay variation. Packet loss has great influence for results (extreme cases are described in the RFC). For reporting and SLA purposes, simplicity is important and PDV lends itself better (e.g., percentiles, median, mean, etc.). PDV metrics can also be used with different stream characteristics, such as Poisson streams [RFC3393] and periodic streams [RFC3432], depending on the purpose and testing environment.

6.0 Test Cases

A summary of test cases is shown in Table 6–1. .

Table 6–1. Summary of Test Cases

Ref.	Test Description	Metrics	Packet Size (Bytes)	Test Duration	Flows per Port in Both Directions
Host (PHY-PHY)					
7.1	L3 Fwd (no pkt modification) 4 ports	Throughput Latency (avg)	64, 128, 256, 512, 256, 512, 1024, 1280, 1518	60 sec	One flow/port in both directions
7.1	L2 Fwd (with pkt modification) 4 ports	Throughput Latency (avg)	64, 128, 256, 512, 256, 512, 1024, 1280, 1518	60 sec	One flow/port in both directions
vSwitch (PHY-OVS-PHY)					
7.2	L3 Fwd 4 ports	Throughput Latency (avg)	64, 128, 256, 512, 256, 512, 1024, 1280, 1518	60 sec	One flow/port in both directions 1000 flows/port in both directions 2000 flows/port in both directions
7.3	L3 Fwd 4 ports Maximum load (from test-case 3) for zero packet loss. Core configuration chosen from test-case 3.	Packet Delay Variation	64B	1 hr	One flow/port in both directions
One VM (PHY-VM-PHY)					
7.4	Single VM (vhost-user) L3 Fwd 4 ports	Throughput Latency (min, max, avg)	64, 128, 256, 512, 256, 512, 1024, 1280, 1518	60 sec	One flow/port in both directions 1000 flows/port in both directions
Two VMs (PHY-VM-VM-PHY)					
7.5	Two VMs in series (vhost-user) L3 Fwd 2 ports	Throughput Latency (min, max, avg)	64, 128, 256, 512, 256, 512, 1024, 1280, 1518	60 sec	One flow/port in both directions 2000 flows/port in both directions

7.0 Test Results

7.1 L2 and L3 Host (PHY-PHY)

The test setup for the host is shown in Figure 7–1.

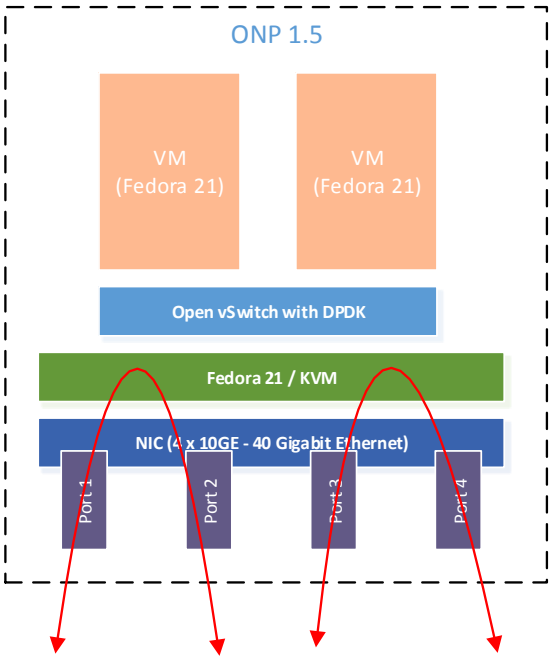


Figure 7–1. Host Setup (PHY-PHY)

As shown in Table 7–1, host tests attempt to achieve system throughput of 40Gbps, using the 4-port configuration with 4 physical cores.

Table 7–1. Host Test Configurations

Test	Configuration Variable				
	Ports	Tx/Rx Queues per Core	Flows per Port in Each Direction	Physical Cores	Hyper-Threaded Cores
L2 Forwarding	4	1	1	4	—
L3 Forwarding	4	1	1	4	—

For **L2 tests**, the full-line rate is achieved for all packet sizes as shown in the results in Table 7–2.

Table 7–2. L2 Test Results for Host

Packet Size	L2 Forwarding — Bidirectional			
	Throughput Achieved with Zero Packet Loss			Average Latency (ns)
	Mbps	Packets/sec	% Line Rate	
64	28,669	42,662,748	72	14,552
72	31,222	42,420,571	78	13,608
128	40,000	33,783,756	100	37,606
256	40,000	18,115,929	100	36,964
512	40,000	9,398,488	100	39,504
768	40,000	6,345,174	100	43,534
1024	40,000	4,789,270	100	49,052
1280	40,000	3,846,150	100	53,755
1518	40,000	3,250,973	100	58,041
Affinity Details	./l2fwd -c 1e -n 4 --socket-mem 1024,0 -- -p 0f			

For **L3 tests,** the full-line rate is achieved for all packet sizes from 128 bytes as shown in Table 7–3.

Table 7–3. L3 Test Results for Host

Packet Size	L3 Forwarding — Bidirectional			
	Throughput Achieved with Zero Packet Loss			Average Latency (ns)
	Mbps	Packets/sec	% Line Rate	
64	40,000	59,523,774	100	21,831
72	40,000	54,347,790	100	24,791
128	40,000	33,783,750	100	20,681
256	40,000	18,115,928	100	23,083
512	40,000	9,398,484	100	28,781
768	40,000	6,345,173	100	34,447
1024	40,000	4,789,268	100	40,284
1280	40,000	3,846,151	100	46,610
1518	40,000	3,250,971	100	51,822
Affinity Details	./l3fwd -c 1e -n 4 --socket-mem 1024,0 -- -p 0xf --config="(0,0,1),(1,0,2),(2,0,3),(3,0,4)"			

7.2 Virtual Switching (PHY-OVS-PHY) — Throughput

Figure 7–2 shows the test setup for OVS with DPDK for PHY-to-PHY with four 10GbE ports to achieve 40Gbps throughput.

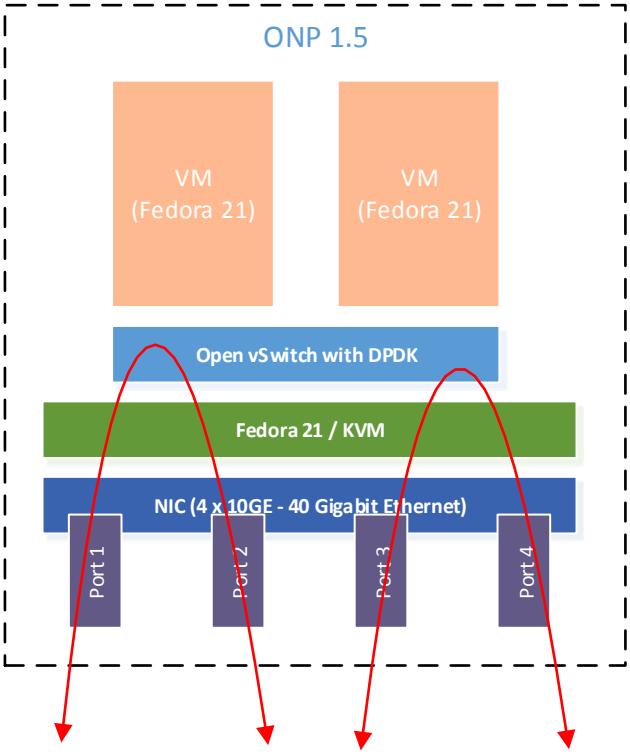


Figure 7–2. Virtual Switching Setup (PHY-OVS-PHY)

Virtual switching tests attempt to achieve system throughput of 40Gbps using 4 ports and 3 configuration variables (see Table 7–4):

- Scaling with cores using 1, 2, or 4 physical cores
- One flow per port (total four flows) compared to 2K flows per port (total 8K flows)
- Hyper-threading turned on or off, comparing 1 physical core vs 2 hyper-threaded cores; and 2 physical cores vs 4 hyper-threaded cores.

Table 7–4. Configurations for Virtual Switching Tests

Test	Configuration Variable				
	Ports	Tx/Rx Queues per Core	Flows per Port in each Direction	Physical Cores	Hyper-Threaded Cores
Core scaling (L3 Forwarding)	4	1	1	1, 2, 4	—
Core scaling (L3 Forwarding)	4	1	2K	1, 2, 4	—
Impact of hyper-threading	4	1	1	1, 2	2, 4
Impact of hyper-threading	4	1	2K	1, 2	2, 4

7.2.1 Core Scaling— One Flow per Port (Total 4 Flows)

Figure 7–3. shows scaling performance with 1, 2, and 4 physical cores using four flows. Maximum theoretical throughput is indicated by the "top purple line" (packets-per-second on the Y-axis).

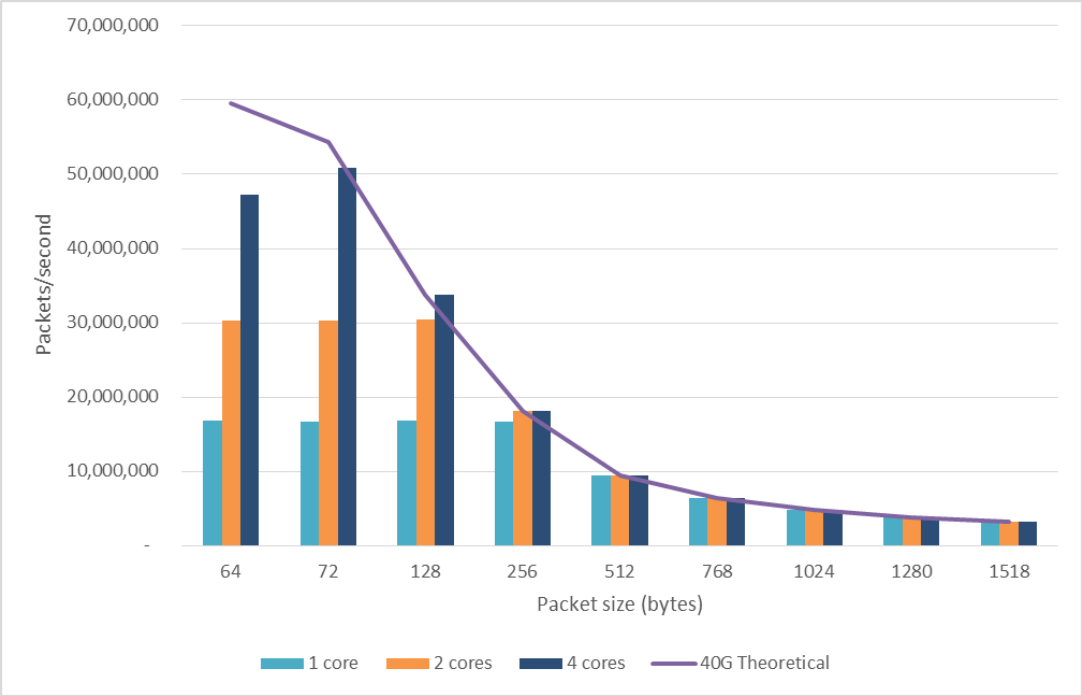


Figure 7–3. OVS with DPDK — Host Throughput with HT Disabled (4 Flows)

The test data in Table 7–5 shows the smallest packet size that achieves line rate when using 1, 2, and 4 physical cores, respectively.

Table 7–5. Packet Sizes that Achieve Line Rate Using 1, 2, and 4 Cores (with 4 Flows)

No. of Physical Cores	Packet Size (Bytes)		
	128B	256B	512B
1 Physical Core Throughput (packets/sec)	16,766,876 (49% of line rate)	16,749,871 (92% of line rate)	9,398,497 (100% of line-rate)
2 Physical Cores Throughput (packets/sec)	30,387,023 (90% of line rate)	18,115,940 (100% of line-rate)	9,398,496 (100% of line rate)
4 Physical Cores Throughput (packets/sec)	33,783,781 (100% of line-rate)	18,115,939 (100% of line rate)	9,398,490 (100% of line rate)

7.2.2 Core Scaling— 2K Flows per Port (Total 8K Flows)

Figure 7–4 shows scaling performance with 1, 2, and 4 physical cores, using 8K flows. Maximum theoretical throughput is indicated by the top purple line (packets-per-second on the Y-axis).

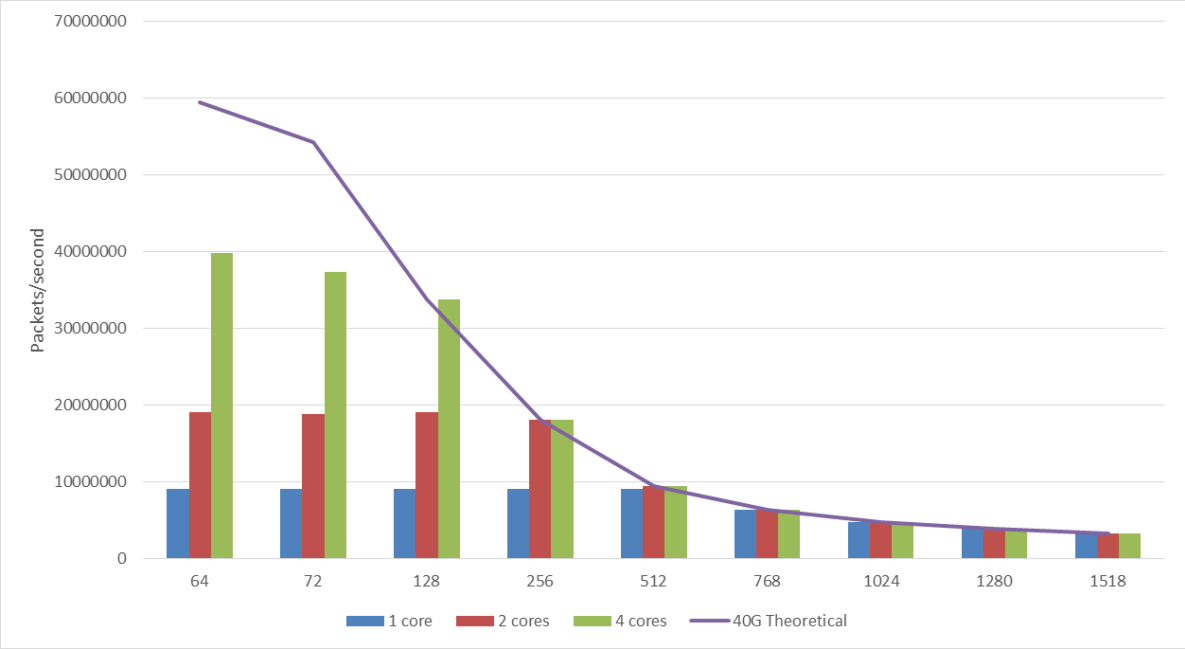


Figure 7–4. OVS with DPDK — Host Throughput with HT Disabled (8K flows)

The test data in

shows the smallest packet size that achieves line rate when using 1, 2, and 4 physical cores, respectively.

Table 7–6. Packet Sizes that Achieve Line Rate Using 1, 2, and 4 Cores (with 8K Flows)

No. of Physical Cores	Packet Size (Bytes)		
	128B	256B	768B
1 Physical Core Throughput (packets/sec)	9,058,613 (27% of line rate)	9,043,478 (50% of line rate)	6,345,179 (100% of line-rate)
2 Physical Cores Throughput (packets/sec)	19,085,895 (56% of line rate)	18,115,935 (100% of line-rate)	6,345,178 (100% of line rate)
4 Physical Cores Throughput (packets/sec)	18,115,935 (100% of line-rate)	18,115,941 (100% of line rate)	6,345,177 (100% of line rate)

7.2.3 64-Byte Performance

7.2.3.1 Core Scalability for 64B Packets

Figure 7–5 shows scaling performance of 64-byte packets with 1, 2, and 4 physical cores with the following number of flows configured:

- 4 total flows (1 flow per port)
- 4K total flows (1K flows per port)
- 8K total flows (2K flows per port)

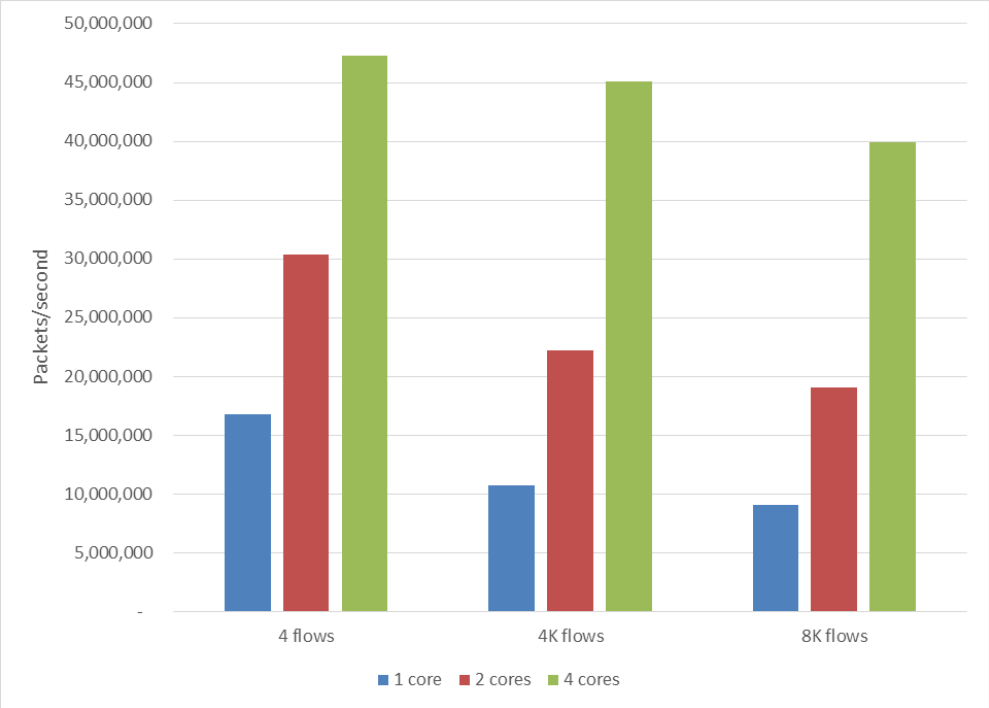


Figure 7–5. Core Scaling for 64B Packets with 4 Flows and 4K and 8K Flows

Test data for measured throughput for 64B packets is shown in Table 7–7. This shows fairly linear scaling when using 1, 2, or 4 cores up to 8K flows. Due to the current configuration of OVS hash lookup tables, significant degradation in performance is observed when using more than 8K flows. This is related to the size of the EMC (exact match cache), which is a hash table in OVS. The current size of the EMC is set to 8K (flows) by default. Using this default configuration, larger numbers of flows may use a slower data path (not the EMC).

In Table 7–7, there is ~46% performance drop from 4 flows to 8K flows for 1 physical core, while 4K flows show ~36% performance drop compared to 4 flows for 1 physical core.

Table 7–7. 64B Performance Scaling with 1, 2, and 4 Cores

Number of Flows	1 Physical Core Measured throughput		2 Physical Cores Measured throughput		4 Physical Cores Measured throughput	
	Packets/sec	Line Rate %	Packets/sec	Line Rate %	Packets/sec	Line Rate %
4	16,766,108	28	30,347,400	51	47,266,489	79
4k	10,781,154	18	22,233,135	37	45,079,543	75
8k	9,054,720	15	19,067,981	32	39,900,258	67

7.2.3.2 Performance with Hyper-threading

Figure 7–6 shows that hyper-threading increases performance of one 64-byte flow by 14% when one core is used and 17% when two cores are used.

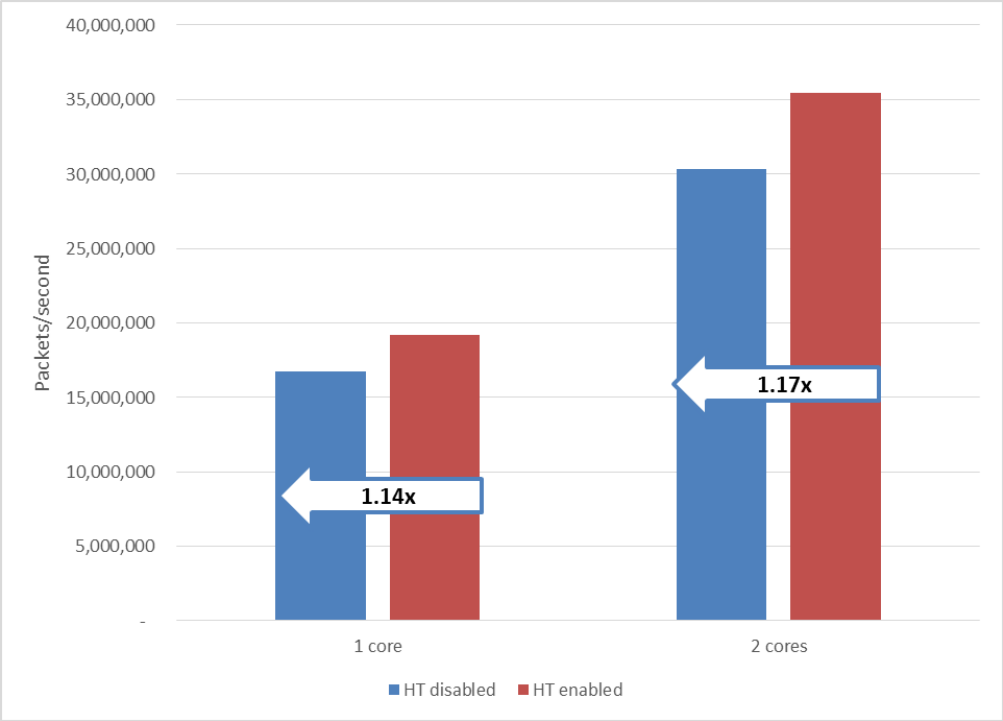


Figure 7–6. Performance Scaling with Hyper-Threading Using 4 Flows (1 Flow per Port)

Figure 7–7 shows hyper-threading increases performance of 4K, 64-byte flows by 49% when 1 core is used and 37% when 2 cores are used.

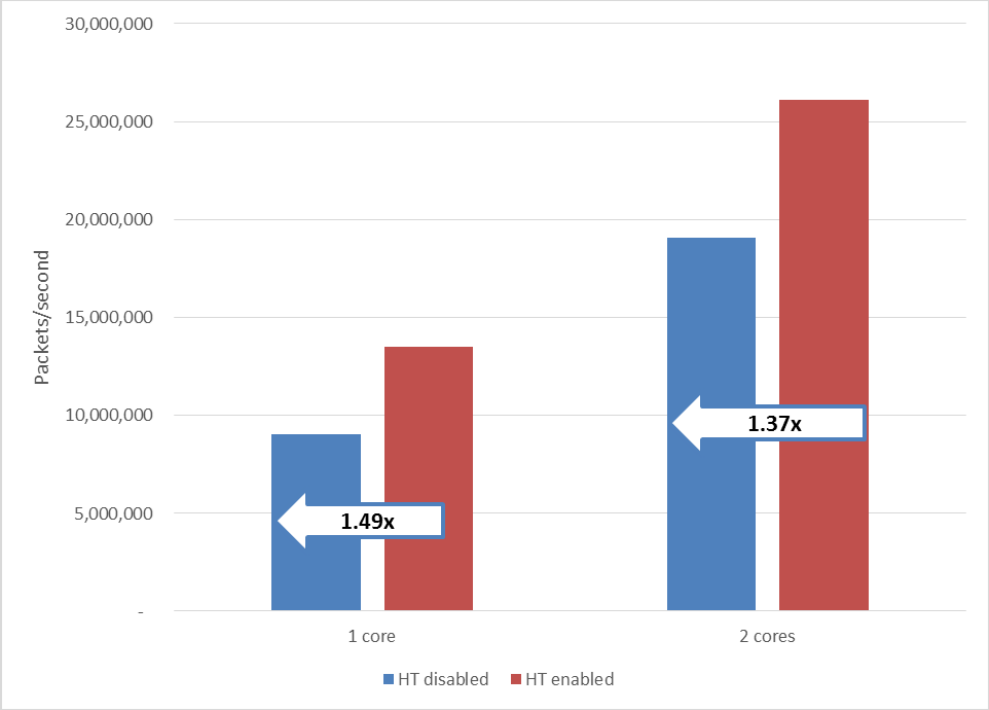


Figure 7–7. Performance Scaling with Hyper-Threading with 8K Flows (2K Flows per Port)

The test data in Table 7–8 shows the measured throughput for 64B packets with 4-flow and 8K-flow configurations.

Table 7–8. 64B Performance with and without Hyper-Threading for 4 Flows and 8K Flows

Number of Flows	1 Physical Core Throughput (packets/sec)	2 Hyper-threaded Cores Throughput (packets/sec)	2 Physical Cores Throughput (packets/sec)	4 Hyper-threaded Cores Throughput (packets/sec)
4	16,766,108 pps (28% of maximum)	19,183,077 pps (32% of maximum)	30,347,400 pps (51% of aximum)	35,469,109 pps (60% of maximum)
8K	9,054,720 pps (15% of maximum)	13,485,897 pps (23% of maximum)	19,067,981 pps (32% of maximum)	26,088,780 pps (44% of maximum)

7.3 Virtual Switching (PHY-OVS-PHY) — Latency

As described in section 5.4, RFC 5481 specifies two forms of packet delay variation (PDV). The Ixia packet generator used does not support RFC 4581 for PDV or the measurement of IPDV. The generator does provide a packet delay variation measurement with three modes (FIFO, LILO, and FILO). The RFC 5481 metrics of PDV and IPDV, however, can be computed off-line, using data capture and post-processing. Techniques and tools for doing this will be discussed in a future Intel® Open Network Platform test document.

Figure 7–8 shows the DUT configuration used for measuring latency through the vSwitch.

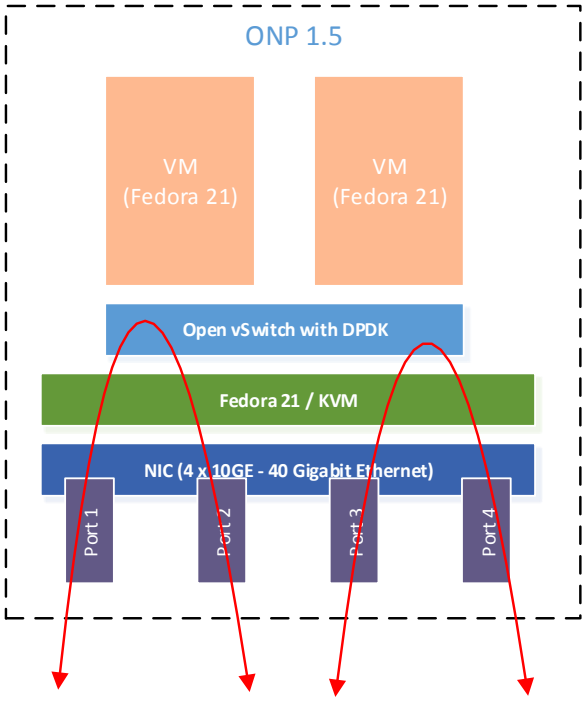


Figure 7–8. Virtual Switching Setup for Measuring Latency (PHY-OVS-PHY)

Figure 7–9 shows PDV for 64B packets during a one hour test for each target load. The latency values represent the aggregate of latency measurements for all flows during the test run (i.e., for each target load). The target load is increased from 10% of line rate up to 80% of line-rate in 10% increments. The target load of 80% represents the maximum load without packet loss as determined by test case three (Virtual Switching Tests — Throughput).

To illustrate the results, the following is extrapolated from the graph:

- Target load is 60% of line rate (orange bars).
- Approximately 30% of packets during a one hour test have latency between 4 and 5.7 microseconds.
- Approximately 70% of packets during a one hour test have latency between 5.7 and 8 microseconds.
- Total number of packets is 100% of line rate.

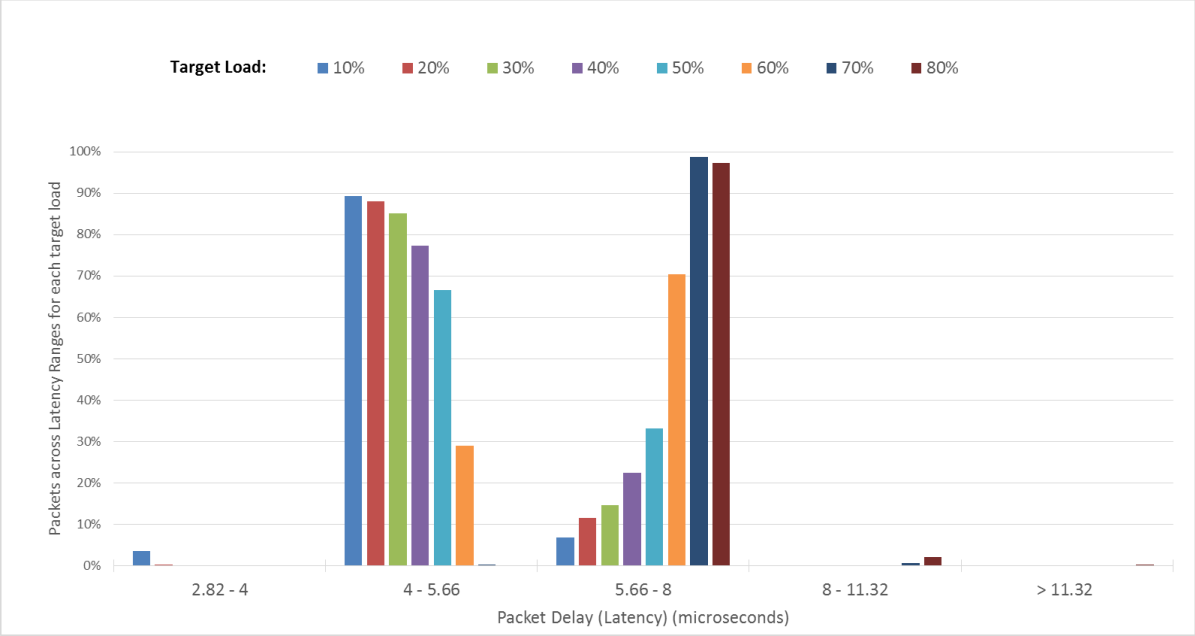


Figure 7–9. Packet Delay Variation with Varying Load (64B Packets, Zero Packet Loss)

7.4 One VM (PHY-VM-PHY)

One VM test is set up with two bidirectional flows (total four flows) using 4 ports as shown in the Figure 7–10. One-VM Setup (

Note: PHY-to-VM tests are recommended to set to a single core. Further performance analysis for multiple cores is under study.

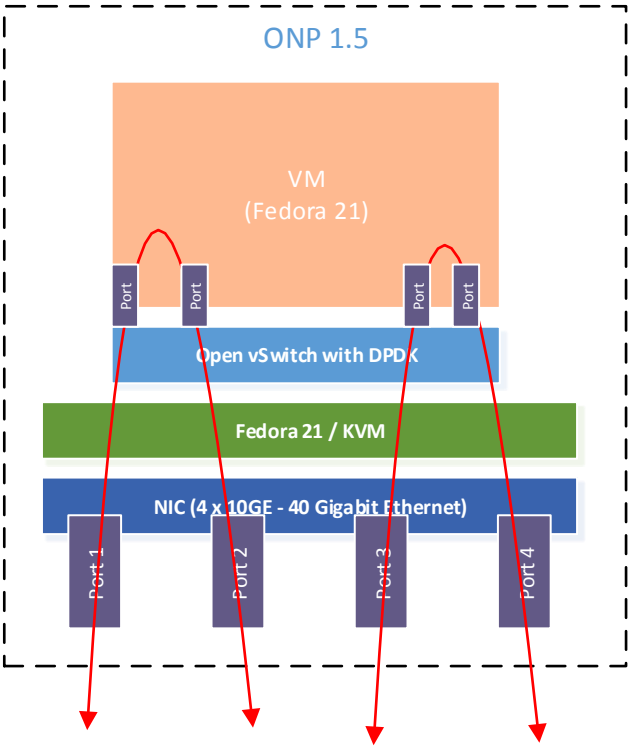


Figure 7–10. One-VM Setup (PHY-VM-PHY)

Note: Two switching operations take place while packets are being routed through the system.

Table 7–9 and Table 7–10 show the L3fwd performance using 1 core running 4 flows and 4K flows with 1 core running a 1 OVS PMD thread.

Table 7–9. Packet Throughput with 4 Flows and 1 Core

Packet size (Bytes)	Aggregate Throughput (Packets/sec)	Line Rate %	Minimum Latency (µs)	Average Latency (µs)	Maximum Latency (µs)
64	4,796,202	8	13.7	57.2	892
72	4,746,944	9	9.3	97.6	1,638
128	4,681,890	14	11.4	59.0	708
256	4,367,109	24	9.4	43.1	157
512	4,064,758	43	12.9	104.9	1,420
768	2,915,991	46	12.7	133.7	2,198
1024	2,386,169	50	12.8	122.2	2,154
1280	2,076,170	54	14.8	99.8	908
1518	1,852,326	57	10.7	34.3	282

Table 7–10. Packet Throughput with 4K Flows and 1 Core

Packet size (Bytes)	Aggregate Throughput (Packets/sec)	Line Rate %	Minimum Latency (µs)	Average Latency (µs)	Maximum Latency (µs)
64	3,069,777	5	8.2	73.7	1,003
72	3,013,002	6	10.6	92.1	1,713
128	3,016,125	9	11.0	87.2	1,567
256	2,948,439	16	8.7	105.6	1,760
512	2,738,137	29	10.5	106.1	1,701
768	1,977,414	31	12.6	557.6	3,835
1024	1,608,287	34	12.0	94.0	2,166
1280	1,455,189	38	11.0	155.3	3,173
1518	1,318,009	41	7.0	88.7	1,613

Figure 7–11 shows the L3 forwarding throughput performance of 4 flows and 4K flows with 1 core (without hyper-threading) with 40Gbps throughput rate. There is an average of 33% performance decrease for 4K flows in comparison to 4 flows.

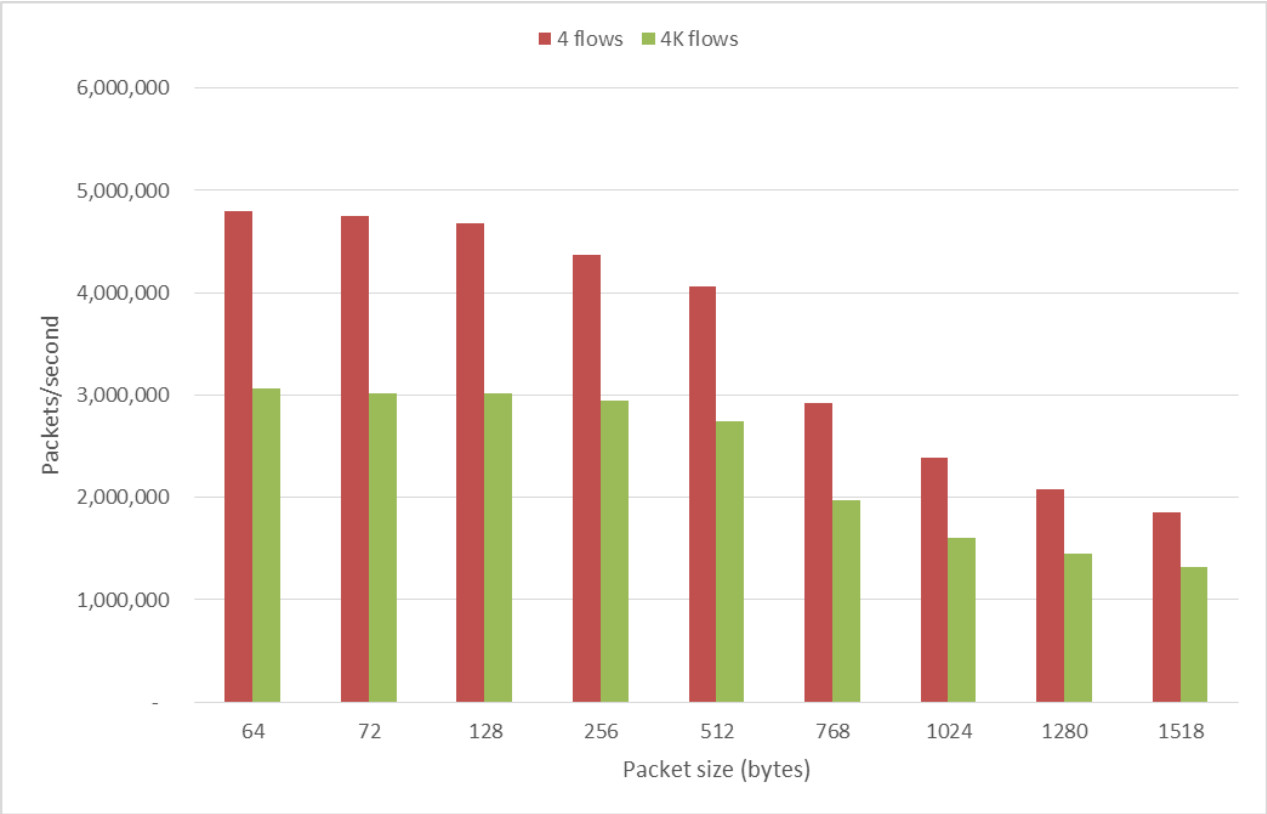


Figure 7–11. One-VM Throughput (PHY-VM-PHY) with 4 Flows and 4K Flows

7.5 Two VMs (PHY-VM-VM-PHY)

Figure 7–12 shows the VM-VM test setup with 2 x 10GbE ports (20Gbps) with packets being forwarded from the first VM to the second VM (total two flows).

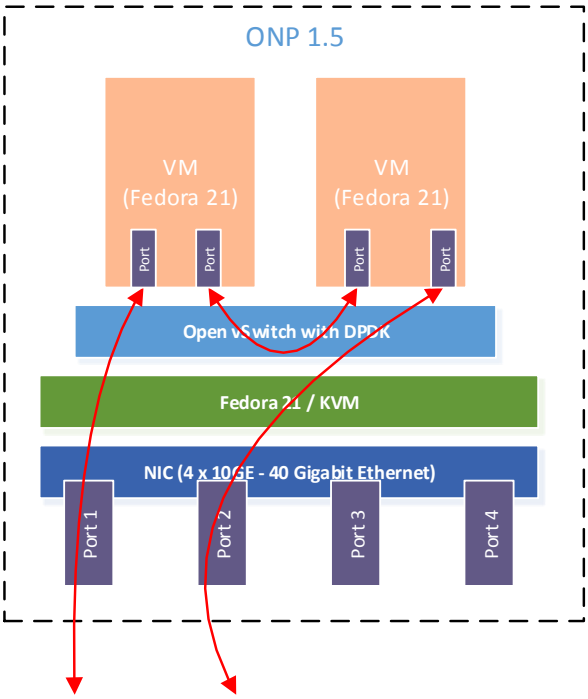


Figure 7–12. Two-VM Setup (PHY-VM-VM-PHY)

Note: There are 3 switching operations taking place while packets are being routed through the system.

Figure 7–13 shows packet throughput comparing 2 flows and 4K flows for 1 core running 1 OVS PMD thread.

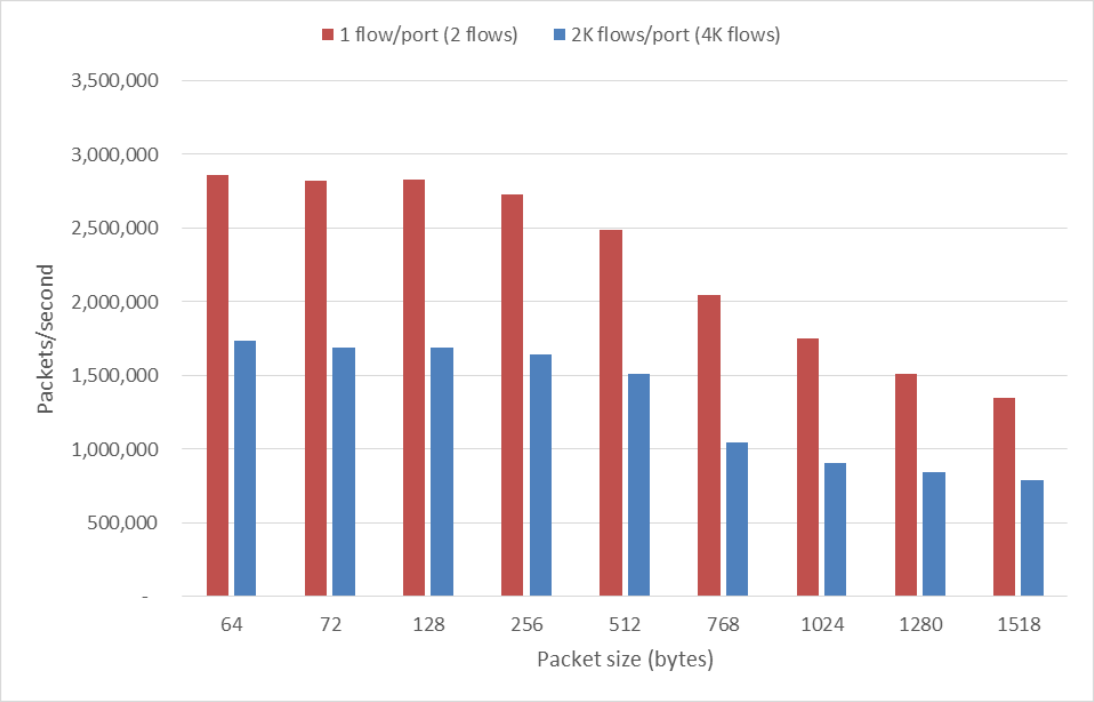


Figure 7–13. Two-VM Throughput (PHY-VM-VM-PHY) with 2 Flows and 4K Flows

Table 7–11 and Table 7–12 show the data plotted and the latency numbers for each packet size for 1 core running 1 OVS PMD thread.

Table 7–11. VM-to-VM Packet Throughput with 2 Flows with 1 Core

Packet size (Bytes)	Aggregate throughput (packets/sec)	Line Rate %	Minimum latency (µs)	Average latency (µs)	Maximum latency (µs)
64	2,858,482	9.6	11	72	1,009
72	2,820,089	10.4	12	125	1,288
128	2,830,877	16.8	18	210	1,504
256	2,726,504	30.1	12	144	1,225
512	2,486,700	52.9	15	218	1,454
768	2,046,907	64.5	20	141	1,210
1024	1,751,030	73.1	20	131	984
1280	1,512,187	78.6	24	151	1,147
1518	1,344,186	82.7	23	116	1,062

Table 7–12. VM-to-VM Packet Throughput with 4K Flows with 1 Core

Packet size (Bytes)	Aggregate throughput (packets/sec)	Line Rate %	Minimum latency (µs)	Average latency (µs)	Maximum latency (µs)
64	1,736,304	5.8	17	191	2,257
72	1,690,405	6.2	15	133	1,139
128	1,687,704	10.0	19	122	920
256	1,640,607	18.1	15	144	1,020
512	1,509,908	32.1	17	121	601
768	1,043,917	32.9	19	242	1,697
1024	903,695	37.7	12	32	1,554
1280	841,002	43.7	19	316	2,291
1518	787,867	48.5	12	275	1,782

Figure 7–14 compares packet throughput of PHY-to-VM case and VM-to-VM test case with 4k flows (without hyper-threading) for 1 core running 1 OVS PMD thread. There is an additional switching operation in the VM-to-VM setup for communicating between the two VMs.

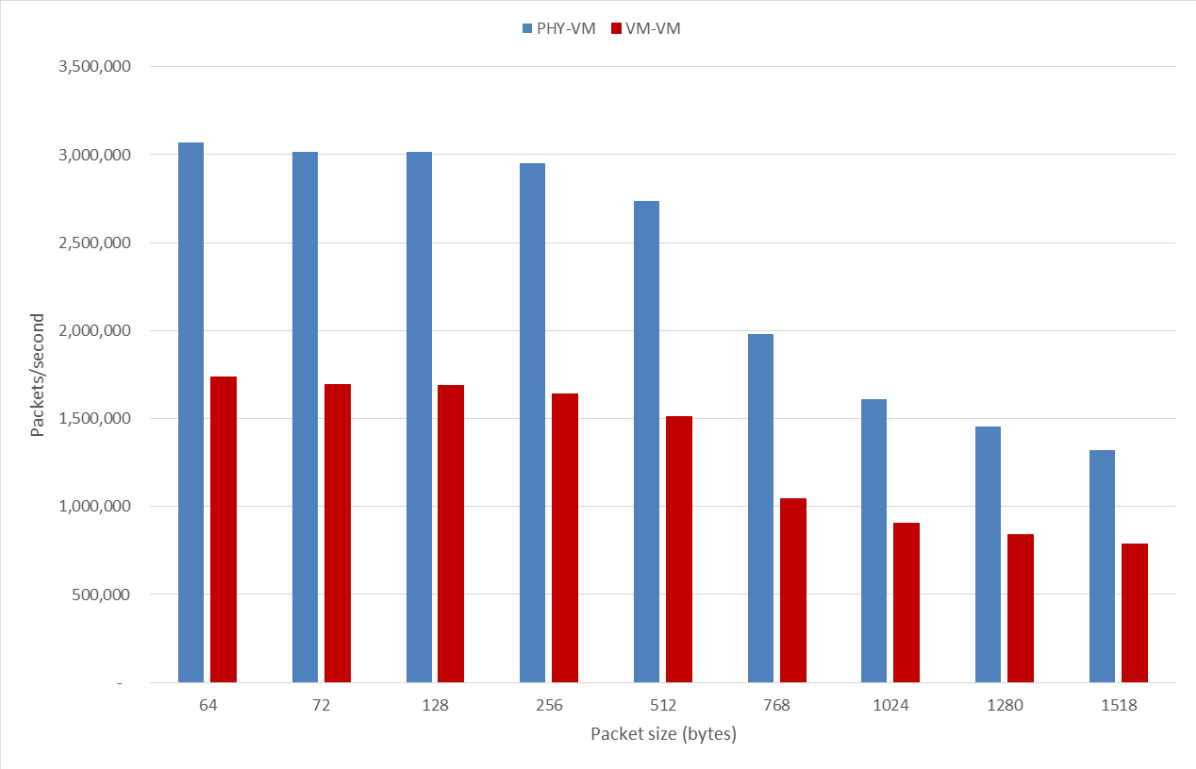


Figure 7–14. Throughput Comparison Using 1 and 2 VM Configurations with 4K Flows (1 Core)

8.0 Industry Benchmarks

8.1 ETSI NFV

The ETSI NFV (Phase II) is developing test methodologies and test specifications relevant to performance testing. Certain draft specification documents are available publically here: https://docbox.etsi.org/ISG/NFV/Open/Drafts/. This includes a "NFV Pre-deployment Validation" specification with the following:

1. Test methods for pre-deployment validation:
    a. Validating physical DUTs and SUTs:
        i. Data plane validation
        ii. Control plane validation
        iii. Management plane validation
    b. Impact of virtualization on test methods
    c. Considerations on choice of virtualized versus hardware based test appliances
2. Pre-deployment validation of NFV infrastructure
3. Pre-deployment validation of VNFs:
    a. VNF life-cycle testing:
        i. VNF instantiation testing
        ii. VNF termination
    b. VNF data plane benchmarking
4. Pre-deployment validation of network services
5. Reliability & resiliency requirements
6. Security considerations

8.2 IETF

The Benchmark Working Group (BMWG) is one of the longest-running working groups in IETF. This group was rechartered in 2014 to include benchmarking for virtualized network function (VNF) and their infrastructure.

An active Internet draft, "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure," is available here: https://tools.ietf.org/html/draft-ietf-bmwg-virtual-net-00. Many RFCs referenced originated in the BMWG, including foundational RFC 1242 and RFC 2544:

- RFC 1242 Benchmarking Terminology for Network Interconnection Devices
- RFC 2544 Benchmarking Methodology for Network Interconnect Devices
- RFC 2285 Benchmarking Terminology for LAN Switching Devices
- RFC 2889 Benchmarking Methodology for LAN Switching Devices

- RFC 3918 Methodology for IP Multicast Benchmarking
- RFC 4737 Packet Reordering Metrics
- RFC 5481 Packet Delay Variation Applicability Statement
- RFC 6201 Device Reset Characterization

8.3 Open Platform for NFV (OPNFV)

OPNFV is a carrier-grade, integrated, open-source platform to accelerate the introduction of new NFV products and services. As an open-source project, OPNFV is uniquely positioned to bring together the work of standards bodies, open-source communities, and commercial suppliers to deliver a de facto open-source NFV platform for the industry. By integrating components from upstream projects, the community can conduct performance and use case-based testing to ensure the platform's suitability for NFV use cases.

As shown in Figure 8–1, many test projects within OPNFV are concerned with performance.

Base system testing:

- Infrastructure verification
- Platform performance benchmarking
- Characterize vSwitch performance for Telco
- Find system bottlenecks
- Storage performance benchmarking for NFVI
- Carrier grade requirements
- Controller performance testing

For more information, refer to OPNFV WiKi: https://wiki.opnfv.org/start.

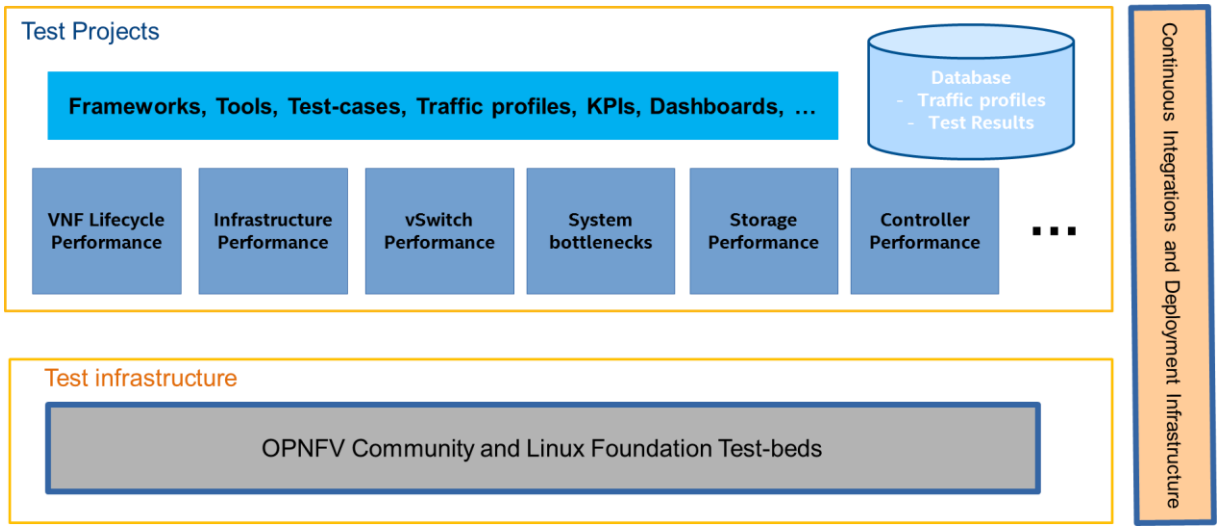


Figure 8–1. OPNFV Test Infrastructure

vSwitch performance characterization is of particular relevance to this test report. An Internet draft for benchmarking virtual switches in OPNFV is available here: https://tools.ietf.org/html/draft-vsperf-bmwg-vswitch-opnfv-00. The draft describes the progress of the OPNFV project on virtual switch performance. This project intends to build on the current and completed work of the BMWG in IETF. The BMWG has traditionally conducted laboratory characterization of dedicated physical implementations of Internet-working functions. This memo begins to describe the additional considerations when vSwitches are implemented in general-purpose hardware.

9.0 Performance Tuning

9.1 Tuning Methods

There are a few important tuning methods that can improve throughput performance for PHY-PHY, PHY-VM, and VM-VM test cases:

- CPU core isolation for OVS-DPDK
- HugePage size 1 GB
- CPU core affinity for ovs-vswitchd and OVS PMD threads
- CPU core affinity for the VM (qemu-kvm)

This section provides some fundamental optimization and tunings for the OVS with DPDK setup. Refer to https://github.com/openvswitch/ovs/blob/master/INSTALL.DPDK.md#performance-tuning for more information on tuning-related optimization.

9.2 CPU Core Isolation for OVS-DPDK

While the threads used by OVS are pinned to logical cores on the system, the Linux scheduler can also run other tasks on those cores. To help prevent additional workloads from running on them, the isolcpus Linux* kernel parameter can be used to isolate the cores from the general Linux scheduler. Add the isolcpus Linux* parameter in the Linux boot kernel of the host machine. For example, if OVS vswitchd and qemu-kvm process are to run on logical cores 2, 4, and 6, the following should be added to the kernel parameter list:

```
isolcpus=2,4,6
```

9.3 HugePage Size 1 GB

HugePage support is required for the large-memory pool allocation used for packet buffers. By using HugePage allocations, performance is increased because fewer pages are needed, and therefore less translation lookaside buffers (TLBs, high-speed translation caches). This reduces the time it takes to translate a virtual page address to a physical page address. Without HugePages, high TLB miss rates would occur with the standard 4K page size, slowing performance.

The allocation of HugePages should be done at boot time or as soon as possible after system boot to prevent memory from being fragmented in physical memory. To reserve HugePages at boot time, a parameter is passed to the Linux* kernel on the kernel command line. For example, to reserve 16G of HugePage memory in the form of 16 1G pages, the following options should be passed to the kernel:

```
default_hugepagesz=1G hugepagesz=1G hugepages=16
```

Note: For 1G HugePages, it is not possible to reserve the HugePage memory after the system has booted.

After the machine is up and running, mount the huge table file system:

```
mount -t hugetlbfs -o pagesize=1G none /dev/hugepages
```

9.4 CPU Core Affinity for ovs-vswitchd and OVS PMD Threads

With PMD multi-threading support, OVS creates one PMD thread for each NUMA node as default. The PMD thread handles the I/O of all DPDK interfaces on the same NUMA node. The following command can be used to configure the multi-threading behavior:

```
`ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=<hex string>`
```

The above command asks for a CPU mask for setting the affinity of PMD threads. A set bit in the mask means a PMD thread is created and pinned to the corresponding CPU core. Ideally, for maximum throughput, the PMD thread should not be scheduled out, which temporarily halts its execution. Therefore, with the CPU core isolation being on the host machine during boot time, the CPU-isolated cores will be used to set the affinity of the PMD threads. For example, to configure PMD threads on core 2 and 3 using `'pmd-cpu-mask'`:

```
ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=C
```

Check that the OVS PMD thread is set to the correct CPU1 and ovs-vswitchd threads are set to CPU2 and CPU3 using this command:

```
#  top -p `pidof ovs-vswitchd` -H -d1
```

Sample output:

```
top - 17:31:09 up  2:46,  3 users,  load average:
0.40, 0.11, 0.08

Threads:  18 total,   1 running, 17 sleeping,   0
stopped,   0 zombie

%Cpu(s):  8.4 us, 0.0 sy, 0.0 ni, 91.6 id, 0.0 wa,
0.0 hi, 0.0 si, 0.0 st

KiB Mem: 32748524 total, 11233304 free, 21292684
used, 222536 buff/cache

KiB Swap:  4194300 total, 4194300 free,       0
used. 11237940 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S
%CPU %MEM     TIME+ COMMAND

  2150 root      20   0 3836184   8896   5140 R
99.0  0.0   0:28.55 pmd28

  2152 root      20   0 3836184   8896   5140 R
99.0  0.0   0:28.55 pmd29

  2041 root      20   0 3836184   8896   5140 S
0.0  0.0   0:13.47 ovs-vswitchd

 2042 root      20   0 3836184   8896   5140 S
0.0  0.0   0:00.00 ovs-vswitchd
```

Note: The PMD threads on a NUMA node are only created if there is at least one DPDK interface from the NUMA node that has been added to OVS. To understand where most of the time is spent and whether the caches are effective, these commands can be used:

```
# ovs-appctl dpif-netdev/pmd-stats-clear #To reset statistics
# ovs-appctl dpif-netdev/pmd-stats-show
```

9.5 CPU Core Affinity for the Virtual Machine (qemu-kvm)

When configuring a PHY-VM test environment, it is important to set the CPU core affinity for the virtual machine (VM). Depending on the number of cores being assigned to the VM, the CPU core affinity should be set according to the QEMU threads. For example, to configure a VM with 4 cores, start the VM on CPU 4-6 (0x70):

```
taskset 70 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/vm-
fc21.img -boot c -enable-kvm -pidfile /tmp/vm1.pid -monitor
unix:/tmp/vm1monitor,server,nowait -name 'FC21-VM1' -net none -no-reboot -object
memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -numa
node,memdev=mem -mem-prealloc -net none \
-chardev socket,id=char1,path=/usr/local/var/run/openvswitch/vhost-user0 \
-netdev type=vhost-user,id=net1,chardev=char1,vhostforce -device virtio-net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso6=off
,guest_ecn=off,mrg_rxbuf=off \
-chardev socket,id=char2,path=/usr/local/var/run/openvswitch/vhost-user1 \
-netdev type=vhost-user,id=net2,chardev=char2,vhostforce -device virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso6=off
,guest_ecn=off,mrg_rxbuf=off
--nographic -vnc :14
```

Once the VM is running, there will be multiple QEMU threads that are spawned running on the host. Check the main QEMU thread PID to track the spawned threads:

```
# ps -e |grep qemu
  2511 pts/3    22:27:53 qemu-system-x86
```

Use the `top` command to provide a list of the main and child process QEMU threads. The main QEMU thread PID 2511 is always active with utilization close to 100% of %CPU:

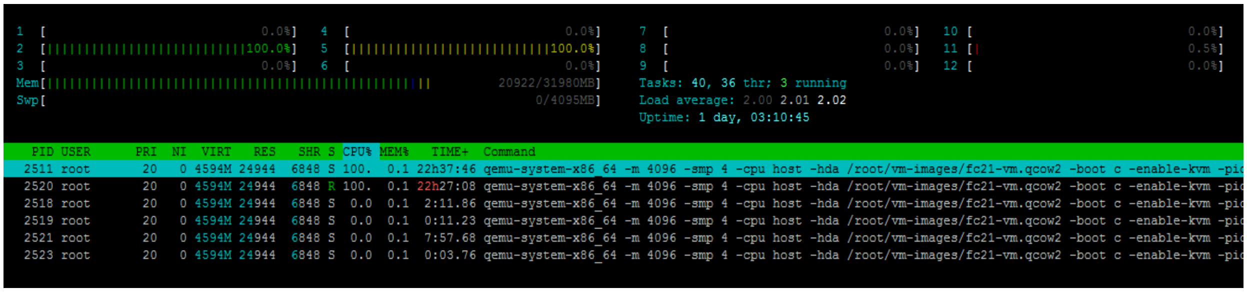
```
# top -p 2511 -H -d1
```

```
top - 17:06:42 up 1 day,  3:03,  3 users,  load average: 2.00, 2.01, 2.02

Threads:   6 total,   1 running,   5 sleeping,   0 stopped,   0 zombie

%Cpu(s): 16.7 us, 0.0 sy, 0.0 ni, 83.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem: 32748524 total, 10566116 free, 21308332 used,  874076 buff/cache

KiB Swap:  4194300 total, 4194300 free,       0 used. 11189840 avail Mem



   PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+
COMMAND

  2520 root      20   0 4704308  24944   6848 R 99.9  0.1   1339:34 qemu-
system-x86

  2511 root      20   0 4704308  24944   6848 S  0.0  0.1   0:11.69 qemu-
system-x86

  2518 root      20   0 4704308  24944   6848 S  0.0  0.1   2:11.77 qemu-
system-x86

  2519 root      20   0 4704308  24944   6848 S  0.0  0.1   0:11.13 qemu-
system-x86

  2521 root      20   0 4704308  24944   6848 S  0.0  0.1   7:57.56 qemu-
system-x86

  2523 root      20   0 4704308  24944   6848 S  0.0  0.1   0:03.76 qemu-
system-x86
```

Then, use `htop` to check the CPU usage% runtime for each QEMU child thread and determine the active QEMU threads:

```
# htop -p 2520,2511,2518,2519,2521,2523
```

Output:



```
  1  [                             0.0%]   4  [                             0.0%]     7  [                             0.0%]   10 [                             0.0%]
  2  [|||||||||||||||||||||||||||100.0%]   5  [|||||||||||||||||||||||||||100.0%]     8  [                             0.0%]   11 [|                            0.5%]
  3  [                             0.0%]   6  [                             0.0%]     9  [                             0.0%]   12 [                             0.0%]
  Mem[||||||||||||||||||||||||||||||||||||||||||||||||||||         20922/31980MB]     Tasks: 40, 36 thr; 3 running
  Swp[                                                                  0/4095MB]     Load average: 2.00 2.01 2.02
                                                                                      Uptime: 1 day, 03:10:45

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2511 root       20   0 4594M 24944  6848 S 100.  0.1 22h37:46 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/fc21-vm.qcow2 -boot c -enable-kvm -pic
 2520 root       20   0 4594M 24944  6848 R 100.  0.1 22h27:08 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/fc21-vm.qcow2 -boot c -enable-kvm -pic
 2518 root       20   0 4594M 24944  6848 S  0.0  0.1  2:11.86 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/fc21-vm.qcow2 -boot c -enable-kvm -pic
 2519 root       20   0 4594M 24944  6848 S  0.0  0.1  0:11.23 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/fc21-vm.qcow2 -boot c -enable-kvm -pic
 2521 root       20   0 4594M 24944  6848 S  0.0  0.1  7:57.68 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/fc21-vm.qcow2 -boot c -enable-kvm -pic
 2523 root       20   0 4594M 24944  6848 S  0.0  0.1  0:03.76 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/fc21-vm.qcow2 -boot c -enable-kvm -pic
```

From the htop output screen, you can view two active QEMU threads that have high CPU usage%. In this example, PID 2511 and PID 2520 (screen output) are using 100 CPU%. We have to set these two active threads to specific CPU logical cores. We are going to set PID 2511 to CPU4 (0x10), and PID 2520 to CPU 5 (0x20). The other 4 threads (PID: 2518, 2519, 2521, 2523) are going to be set to CPU6 (0x40).

It is important to assign each active (100 %CPU) QEMU thread to separate CPU cores to sustain good optimal throughput performance. If the active QEMU threads are not core-affinitized, the overall throughput performance is impacted.

9.6 Troubleshooting Tips for OVS

In the OVS controller, there are a few management tools in ovs-vswitchd that are useful to monitor the status of ports and OpenFlow activities:

- `ovs-vsctl` manages the switch through interaction with ovsdb-server.
- `ovs-ofctl` is a management utility for OpenFlow.
- `ovs-appctl` is a utility for managing logging levels.

After creating and configuring the ports, the `ovs-vsctl` command tool is useful to check the overall view of the bridges and ports created in the ovsdb-server database:

```
# ovs-vsctl show
```

```
7bdd3285-c5db-4944-b963-3ecedf661a41
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
        Port "dpdk0"
            Interface "dpdk0"
                type: dpdk
        Port "dpdk1"
            Interface "dpdk1"
                type: dpdk
```

The `ovs-ofctl` command tool is useful to check the OpenFlow flow configuration and port statistics. To check port information on a particular bridge, such as the port's MAC address and number, ovs-ofctl show `<bridge-name>` or `ovs-ofctl dump-ports-desc <bridge-name>` provides the following information on all ports:

```
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000001b21a272e4
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
 1(dpdk0): addr:00:1b:21:a2:72:e4
     config:     0
     state:      LINK_DOWN
     current:    AUTO_NEG
     speed: 0 Mbps now, 0 Mbps max
 2(vxlan0): addr:c2:7a:99:d6:01:e2
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-int): addr:00:1b:21:a2:72:e4
     config:     0
     state:      0
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

When the test is running, you can monitor packets sending and receiving at the ports configured in OVS by checking the flow and port statistics. For example, if you want to check if the packets are being received and sent in a flow, `ovs-ofctl dump-flows <bridge-name>` prints all the configured flow statistics. The figure below shows the flows configured for sending and receiving exist and are being used with `n_packets` equal to non-zero.

```
# /root/ovs/utilities/ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4): cookie=0x0, duration=177593.242s, table=0,
n_packets=1300667542, n_bytes=78040052520, idle_age=65534, hard_age=65534,
ip,in_port=2 actions=output:1
```

The `ovs-ofctl dump-ports <bridge-name>` command prints port statistics for RX/TX packets, packets dropped, and packet errors (if they occur). In this example, there are packet errors in port 1. One of the reasons may be that the packet rate being received at port 1 is too high and beyond the port's capacity. The packet sending rate to the port, therefore, needs to be reduced to fix the packet error. If there is a packet drop in the OVS, check the CPU core affinitization for the QEMU threads for the PHY-VM test case, and if the HugePage size is set correctly, and the ovs-vswitchd and OVS PMD threads are running on isolated cores.

```
# /root/ovs/utilities/ovs-ofctl dump-ports br-int
OFPST_PORT reply (xid=0x2): 3 ports
  port  2: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=8, bytes=648, drop=0, errs=0, coll=0
  port  1: rx pkts=578932881, bytes=37051704384, drop=0, errs=176810889,
frame=0, over=0, crc=0
           tx pkts=1300667551, bytes=83242723450, drop=0, errs=0, coll=0
  port LOCAL: rx pkts=14, bytes=1156, drop=0, errs=0, frame=0, over=0,
crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
```

To check the ARP cache content, `ovs-appctl tnl/arp/show` prints the learned MAC address and IP address.

```
# /root/ovs/utilities/ovs-appctl tnl/arp/show
IP                          MAC                 Bridge
===============================================
2.2.2.1          00:1b:21:a2:72:e5   br0
2.2.2.2            00:1b:21:a2:72:e6   br0
```

10.0 Test Setup with OVS

10.1 Configure the Host Machine

1. Disable the following services:
   a. IRQ balance:
   ```
   killall irqbalance
   systemctl stop irqbalance.service
   systemctl disable irqbalance.service
   ```
   b. Firewall and iptables:
   ```
   # systemctl stop firewalld.service
   # systemctl disable firewalld.service
   # systemctl stop iptables.service
   ```
   c. SELinux:
   ```
   [root@localhost ~]# vi /etc/selinux/config
   SELINUX=disabled
   ```
   d. Address space layout randomization:
   ```
   echo "# Disable Address Space Layout Randomization (ASLR)" > /etc/sysctl.d/
   aslr.conf
   echo "kernel.randomize_va_space=0" >> /etc/sysctl.d/aslr.conf
   ```
   e. IPv4 forwarding:
   ```
   echo "# Enable IPv4 Forwarding" > /etc/sysctl.d/ip_forward.conf
   echo "net.ipv4.ip_forward=0" >> /etc/sysctl.d/ip_forward.conf

   systemctl restart systemd-sysctl.service
   cat /proc/sys/kernel/randomize_va_space
   0
   cat /proc/sys/net/ipv4/ip_forward
   0
   ```
2. Set the uncore frequency:
   ```
   # rdmsr -p 0  0x620
   c1e
   # rdmsr -p 14 0x620
   c1e
   # wrmsr -p 0 0x620 0x1e1e
   # wrmsr -p 14 0x620 0x1e1e
   ```
3. Set the PCI configuration:
   ```
   # setpci -s 00:03.0 184.l
   0000000
   ```

```
# setpci -s 00:03.2 184.l
0000000
# setpci -s 00:03.0 184.l=0x1408
# setpci -s 00:03.2 184.l=0x1408
```

4. Remove the following modules:

```
# rmmod ipmi_msghandler
# rmmod ipmi_si
# rmmod ipmi_devintf
```

10.2 Set the Kernel Boot Parameters

1. With hyper-threading enabled, add the following to the kernel boot parameters */etc/default/grub* for 2 sockets:

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=fedora-server/root rd.lvm.lv=fedora-server/swap
default_hugepagesz=1G hugepagesz=1G hugepages=16 hugepagesz=2M hugepages=2048
intel_iommu=off isolcpus=1-13,15-27,29-41,43-55 rhgb quiet"
```

2. With hyper-threading disabled, add the following to the kernel boot parameters */etc/default/grub* for 2 sockets:

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=fedora-server/root rd.lvm.lv=fedora-server/swap
default_hugepagesz=1G hugepagesz=1G hugepages=16 hugepagesz=2M hugepages=2048
intel_iommu=off isolcpus=1-13,15-27 rhgb quiet"
```

3. Save the file and update the GRUB config file:

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

4. Reboot the host machine and check to make sure 1GB and 2MB HugePage sizes are created. You should see 16 1GB HugePages and 2048 2MB HugePages:

```
ls /sys/devices/system/node/node0/hugepages/hugepages-*
```

Output:

```
hugepages-1048576kB/    hugepages-2048kB/
cat /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
16
cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
2048
```

10.3 Compile DPDK 2.0

1. Go to the DPDK-2.0.0 directory and do the following:

```
make install T=x86_64-ivshmem-linuxapp-gcc
cd x86_64-ivshmem-linuxapp-gcc
```

2. Edit the config file (vim .config) and set the configuration options:

```
CONFIG_RTE_BUILD_COMBINE_LIBS=y
CONFIG_RTE_LIBRTE_VHOST=y
```

```
CONFIG_RTE_LIBRTE_VHOST_USER=y
```

3. Save the config file and run `make`:

```
EXTRA_CFLAGS=”-g -Ofast” make -j10
```

10.4 Install OVS

Go to OVS directory and run:

```
./boot.sh
./configure --with-dpdk=/root/dpdk-2.0.0/x86_64-ivshmem-linuxapp-gcc \
CFLAGS=”-Ofast -g”

make 'CFLAGS=-g -Ofast -march=native' -j10
```

10.5 Prepare to Start OVS

1. Mount the 1GB HugePage and 2MB HugePage:

```
mkdir -p /mnt/huge
mkdir -p /mnt/huge_2mb
mount -t hugetlbfs nodev /mnt/huge
mount -t hugetlbfs nodev /mnt/huge_2mb -o pagesize=2MB
```

2. Check that HugePages are mounted:

```
[root@localhost ~]# mount
nodev on /mnt/huge type hugetlbfs (rw,relatime)
nodev on /mnt/huge_2mb type hugetlbfs (rw,relatime,pagesize=2MB)
```

3. Load the modules for OVS:

```
rmmod ixgbe
rmmod igb_uio
rmmod cuse
rmmod fuse
rmmod openvswitch
rmmod uio
rmmod eventfd_link
rmmod ioeventfd
rm -rf /dev/vhost-net
modprobe uio
insmod $DPDK_BUILD/kmod/igb_uio.ko
```

4. Check the PCI ID for the 10GbE NIC ports:

```
[root@localhost ~]# lspci|grep Ethernet
01:00.0 Ethernet controller: Intel Corporation Ethernet Controller X710 for
10GbE SFP+ (rev 01)
```

```
01:00.1 Ethernet controller: Intel Corporation Ethernet Controller X710 for
10GbE SFP+ (rev 01)
02:00.0 Ethernet controller: Intel Corporation Ethernet Controller X710 for
10GbE SFP+ (rev 01)
02:00.1 Ethernet controller: Intel Corporation Ethernet Controller X710 for
10GbE SFP+ (rev 01)
```

10.6 Bind 10GbE NIC Ports to the igb_uio Driver

To create a 4-port configuration:

```
python $DPDK_DIR/tools/dpdk_nic_bind.py --bind=igb_uio 01:00.0
python $DPDK_DIR/tools/dpdk_nic_bind.py --bind=igb_uio 01:00.1
python $DPDK_DIR/tools/dpdk_nic_bind.py --bind=igb_uio 02:00.0
python $DPDK_DIR/tools/dpdk_nic_bind.py --bind=igb_uio 02:00.1
python $DPDK_DIR/tools/dpdk_nic_bind.py -status
```

Output:

Network devices using the DPDK-compatible driver:

```
============================================
0000:01:00.0 'Ethernet Controller X710 for 10GbE SFP+' drv=igb_uio unused=i40e
0000:01:00.1 'Ethernet Controller X710 for 10GbE SFP+' drv=igb_uio unused=i40e
0000:02:00.0 'Ethernet Controller X710 for 10GbE SFP+' drv=igb_uio unused=i40e
0000:02:00.1 'Ethernet Controller X710 for 10GbE SFP+' drv=igb_uio unused=i40e
```

Network devices using the kernel driver:

```
====================================
0000:00:19.0 'Ethernet Connection I217-LM' if=enp0s25 drv=e1000e unused=igb_uio
*Active*
0000:05:00.0 'I210 Gigabit Network Connection' if=enp5s0 drv=igb unused=igb_uio
```

Other network devices:

```
=====================
<none>
```

10.7 Remove and Terminate Previous-Run OVS and Prepare

```
pkill -9 ovs
rm -rf /usr/local/var/run/openvswitch
rm -rf /usr/local/etc/openvswitch/
rm -f /tmp/conf.db

mkdir -p /usr/local/etc/openvswitch
mkdir -p /usr/local/var/run/openvswitch
```

10.8 Initialize the New OVS Database

```
export OVS_DIR=/root/OVS/ovs
cd $OVS_DIR
./ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db
./vswitchd/vswitch.ovsschema

#start database server
./ovsdb/ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
                 --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
                 --pidfile --detach

#initialize OVS database
./utilities/ovs-vsctl --no-wait init
```

10.9 Start OVS-vSwitchd

```
#start OVS with DPDK portion using 2GB on CPU2 (0x2):
./vswitchd/ovs-vswitchd --dpdk -c 0x2 -n 4 --socket-mem 2048 --
unix:/usr/local/var/run/openvswitch/db.sock  --pidfile
```

10.10 Tune OVS-vswitchd

You can check the thread siblings list (when hyperthreading is enabled) with the following:

```
cat /sys/devices/system/cpu/cpuN/topology/thread_siblings_list
```

Based on the core thread siblings, you can set/check the PMD mask so that the multiple logical cores are on the same physical core.

1 PMD Configuration

Set the default OVS PMD thread usage to CPU2 (0x4):

```
./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=4
./ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

2 PMD Configuration

For **1** physical core, 2 logical cores (2 PMDs) on a system with HT enabled, check the thread siblings:

```
cat /sys/devices/system/cpu/cpu1/topology/thread_siblings_list
2,30
```

Then set the pmd-cpu-mask to CPU2 and CPU30 (0x40000004):

```
./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=40000004
./ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

For **2** physical cores and 2 logical cores (2 PMDs) on system HT disabled, set the default OVS PMD thread usage to CPU2 and CPU3 (0xC):

```
./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=C
./ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

4 PMD Configuration

For **2** physical cores, 2 logical cores (4PMDs) on system with HT enabled, check the thread siblings:

```
cat /sys/devices/system/cpu/cpu2/topology/thread_siblings_list
2,30
cat /sys/devices/system/cpu/cpu3/topology/thread_siblings_list
3,31
```

Then set the pmd-cpu-mask to CPU2, CPU3, CPU30 and CPU31 (0x C000000C).

```
./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask= C000000C
./ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

For **4** physical cores (4 PMDs) on system HT disabled, set the default OVS PMD thread usage, set default OVS PMD thread usage to CPU2, CPU3, CPU4, CPU5 (0x3C):

```
./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=3C
./ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

10.11 Create the Ports

4-Port Configuration

```
cd /root/ovs
./utilities/ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
./utilities/ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk
./utilities/ovs-vsctl add-port br0 dpdk1 -- set Interface dpdk1 type=dpdk
./utilities/ovs-vsctl add-port br0 dpdk2 -- set Interface dpdk2 type=dpdk
./utilities/ovs-vsctl add-port br0 dpdk3 -- set Interface dpdk3 type=dpdk
./utilities/ovs-vsctl show
```

10.12 Add the Port Flows

```
export OVS_DIR=/root/ovs
cd $OVS_DIR

# Clear current flows
./utilities/ovs-ofctl del-flows br0

#Add flow
./utilities/ovs-ofctl add-flow br0 \
in_port=1,dl_type=0x800,idle_timeout=0,action=output:2
./utilities/ovs-ofctl add-flow br0 \
in_port=2,dl_type=0x800,idle_timeout=0,action=output:1
./utilities/ovs-ofctl add-flow br0 \
in_port=3,dl_type=0x800,idle_timeout=0,action=output:4
./utilities/ovs-ofctl add-flow br0 \
in_port=4,dl_type=0x800,idle_timeout=0,action=output:3
./utilities/ovs-ofctl dump-flows br0
```

11.0 PHY-VM-PHY Test Setup

Follow the steps on the PHY-to-PHY test setup until section 10.10, Tune OVS-vswitchd, and set up 1 core with 1 PMD thread configuration (without hyper-threading) for the PHY-to-VM tests. Follow the instructions in this section to continue on the PHY-to-VM.

11.1 Create the Ports

4-Port configuration

```
cd /root/ovs
./utilities/ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
./utilities/ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk
./utilities/ovs-vsctl add-port br0 dpdk1 -- set Interface dpdk1 type=dpdk
./utilities/ovs-vsctl add-port br0 dpdk2 -- set Interface dpdk2 type=dpdk
./utilities/ovs-vsctl add-port br0 dpdk3 -- set Interface dpdk3 type=dpdk
./utilities/ovs-vsctl add-port br0 vhost-user0 -- set Interface vhost-user0
type=dpdkvhostuser
./utilities/ovs-vsctl add-port br0 vhost-user1 -- set Interface vhost-user1
type=dpdkvhostuser
./utilities/ovs-vsctl add-port br0 vhost-user2 -- set Interface vhost-user2
type=dpdkvhostuser
./utilities/ovs-vsctl add-port br0 vhost-user3 -- set Interface vhost-user3
type=dpdkvhostuser
./utilities/ovs-vsctl show
```

11.2 Add the Port Flows

```
export OVS_DIR=/root/ovs
cd $OVS_DIR

# Clear current flows
./utilities/ovs-ofctl del-flows br0

# Add Flow
./utilities/ovs-ofctl add-flow br0 \
 in_port=1,dl_type=0x800,idle_timeout=0,action=output:5
./utilities/ovs-ofctl add-flow br0 \
 in_port=2,dl_type=0x800,idle_timeout=0,action=output:6
./utilities/ovs-ofctl add-flow br0 \
 in_port=3,dl_type=0x800,idle_timeout=0,action=output:7
./utilities/ovs-ofctl add-flow br0 \
 in_port=4,dl_type=0x800,idle_timeout=0,action=output:8
./utilities/ovs-ofctl add-flows br0 \
```

```
 in_port=5,dl_type=0x800,idle_timeout=0,action=output:1
./utilities/ovs-ofctl add-flow br0 \
 in_port=6,dl_type=0x800,idle_timeout=0,action=output:2
./utilities/ovs-ofctl add-flow br0 \
 in_port=7,dl_type=0x800,idle_timeout=0,action=output:3
./utilities/ovs-ofctl add-flow br0 \
 in_port=8,dl_type=0x800,idle_timeout=0,action=output:4
./utilities/ovs-ofctl dump-flows br0
```

11.3 Power on the VM

Start the VM on CPU 4, CPU 5, and CPU 6 (0x70) with the following configuration:

```
taskset 70 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/vm-
fc21.img -boot c -enable-kvm -pidfile /tmp/vm1.pid -monitor
unix:/tmp/vm1monitor,server,nowait -name 'FC21-VM1' -net none -no-reboot -object
memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -numa
node,memdev=mem -mem-prealloc  \
-chardev socket,id=char1,path=/usr/local/var/run/openvswitch/vhost-user0 \
-netdev type=vhost-user,id=net1,chardev=char1,vhostforce -device virtio-net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso6=off
,guest_ecn=off,mrg_rxbuf=off \
-chardev socket,id=char2,path=/usr/local/var/run/openvswitch/vhost-user1 \
-netdev type=vhost-user,id=net2,chardev=char2,vhostforce -device virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso6=off
,guest_ecn=off,mrg_rxbuf=off \
-chardev socket,id=char1,path=/usr/local/var/run/openvswitch/vhost-user0 \
-netdev type=vhost-user,id=net1,chardev=char1,vhostforce -device virtio-net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso6=off
,guest_ecn=off,mrg_rxbuf=off \
-chardev socket,id=char2,path=/usr/local/var/run/openvswitch/vhost-user1 \
-netdev type=vhost-user,id=net2,chardev=char2,vhostforce -device virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso6=off
,guest_ecn=off,mrg_rxbuf=off \
--nographic -vnc :14
```

11.4 Set the VM Kernel Boot Parameters

1. Add the following to the kernel boot parameters */etc/default/grub*:

   ```
   GRUB_CMDLINE_LINUX="rd.lvm.lv=fedora-server/root rd.lvm.lv=fedora-server/swap
   default_hugepagesz=1G hugepagesz=1G hugepages=1 hugepagesz=2M hugepages=1024
   isolcpus=1,2 rhgb quiet"
   ```

2. Save the file and update GRUB config file:

   ```
   grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

3. Reboot the VM and check to make sure 1GB and 2MB HugePage sizes are created. You should see one 1GB HugePage and 1024 2MB HugePages:

```
ls /sys/devices/system/node/node0/hugepages/hugepages-*
```

Output:

```
hugepages-1048576kB/    hugepages-2048kB/
cat /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
1
cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
1024
```

11.5 Set up the VM HugePages

Mount the HugePage for 1 GB and 2 MB:

```
mount -t hugetlbfs hugetlbfs /mnt/huge
mount -t hugetlbfs none /mnt/huge_2mb -o pagesize=2MB
```

11.6 Set up DPDK 2.0

1. Download DPDK 2.0.0 and compile it:

```
make install T=x86_64-native-linuxapp-gcc
```

2. Edit the test-pmd apps input and output queue size to 2K for better throughput performance:

```
vi /root/dpdk-2.0.0/app/test-pmd/test-pmd.c

/*
* Configurable number of RX/TX ring descriptors.
*/

#define RTE_TEST_RX_DESC_DEFAULT 2048
#define RTE_TEST_TX_DESC_DEFAULT 2048
```

3. Save and build the test-pmd app:

```
export RTE_SDK=/root/dpdk-2.0.0
export RTE_TARGET=x86_64-native-linuxapp-gcc
make
```

11.7 Set up the vHost Network in the VM

1. Load the UIO kernel module in the VM:

```
modprobe uio
insmod /root/dpdk-2.0.0/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko
```

2. Check the PCI ID for the 10GbE NIC ports:

```
lspci -nn

00:04.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
00:05.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
00:06.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
00:07.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
```

3. Bind the user-side vhost network devices with the igb_uio driver:

```
/root/dpdk-2.0.0/tools/dpdk_nic_bind.py -b igb_uio 00:04.0
/root/dpdk-2.0.0/tools/dpdk_nic_bind.py -b igb_uio 00:05.0
/root/dpdk-2.0.0/tools/dpdk_nic_bind.py -b igb_uio 00:06.0
/root/dpdk-2.0.0/tools/dpdk_nic_bind.py -b igb_uio 00:07.0
/root/dpdk-2.0.0/tools/dpdk_nic_bind.py --status
```

Network devices using DPDK-compatible driver

==========

```
0000:00:04.0 'Virtio network device' drv=igb_uio unused=virtio_pci
0000:00:05.0 'Virtio network device' drv=igb_uio unused=virtio_pci
0000:00:06.0 'Virtio network device' drv=igb_uio unused=virtio_pci
0000:00:07.0 'Virtio network device' drv=igb_uio unused=virtio_pci
```

Network devices using kernel driver

==========

```
<none>
```

11.8 Start the test-pmd Application in the VM

1. Run test-pmd app on vCPU1 and vCPU2 (0x6):

```
cd /root/dpdk-2.0.0/x86_64-native-linuxapp-gcc/build/app/test-pmd
./testpmd -c 0x6 -n 4 -- --burst=32 -i --disable-hw-vlan --txd=2048 --rxd=2048 \
--txqflags=0xf00
```

2. In the application, enter fwd and mac_retry commands:

```
testpmd> set fwd mac_retry
```

3. Set the `mac_retry` packet forwarding mode.
4. Start the PMD forwarding operation:

```
testpmd> start
mac_retry packet forwarding - CRC stripping disabled - packets/burst=32
nb forwarding cores=1 - nb forwarding ports=2
RX queues=1 - RX desc=2048 - RX free threshold=32
RX threshold registers: pthresh=8 hthresh=8 wthresh=0
TX queues=1 - TX desc=2048 - TX free threshold=0
TX threshold registers: pthresh=32 hthresh=0 wthresh=0
TX RS bit threshold=0 - TXQ flags=0xf00
```

11.9 CPU Affinity Tuning

The tables below show the host's CPU core affinity settings for PHY-to-VM test configuration for 1 physical core (no hyper-threading). When the VM starts, there are multiple QEMU threads spawned. Refer to section 9.1.4, CPU Core Affinity for the Virtual Machine (qemu-kvm), to set the active QEMU threads to the correct core affinity.

CPU Affinity Setting on the Host

Logical Core	Process
1	ovs-vswitchd
2	PMD0
4, 5, 6	QEMU

QEMU Threads CPU Affinity

Logical Core	Process	CPU% (from htop)
4	QEMU (main thread)	100
5	QEMU	100
6	QEMU	0
6	QEMU	0
6	QEMU	0
6	QEMU	0

Note: Two active threads (with 100 CPU%) are set to 2 different logical cores

12.0 VM-VM Test Setup

Refer to section, 11.0. PHY-VM-PHY Test Setup, to set up the host configurations until section 10.10, Tune OVS-vswitchd, and then set up 1 core with 1 PMD thread configuration (without hyper-threading) for 2 VMs series tests. Follow the instructions below to continue on the VM-to-VM setup.

12.1 Create the Ports

```
cd /root/ovs
./utilities/ovs-vsctl show
./utilities/ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
./utilities/ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk
./utilities/ovs-vsctl add-port br0 dpdk1 -- set Interface dpdk1 type=dpdk
./utilities/ovs-vsctl add-port br0 vhost-user0 -- set Interface vhost-user0
type=dpdkvhostuser
./utilities/ovs-vsctl add-port br0 vhost-user1 -- set Interface vhost-user1
type=dpdkvhostuser
./utilities/ovs-vsctl add-port br0 vhost-user2 -- set Interface vhost-user2
type=dpdkvhostuser
./utilities/ovs-vsctl add-port br0 vhost-user3 -- set Interface vhost-user3
type=dpdkvhostuser

./utilities/ovs-vsctl show
```

12.2 Add the Port Flows

```
export OVS_DIR=/root/ovs
cd $OVS_DIR

# Clear current flows
./utilities/ovs-ofctl del-flows br0

# Add Flow
./utilities/ovs-ofctl add-flow br0 \
 in_port=1,dl_type=0x800,idle_timeout=0,action=output:3
./utilities/ovs-ofctl add-flow br0 \
 in_port=2,dl_type=0x800,idle_timeout=0,action=output:6
./utilities/ovs-ofctl add-flow br0 \
 in_port=3,dl_type=0x800,idle_timeout=0,action=output:1
./utilities/ovs-ofctl add-flow br0 \
 in_port=4,dl_type=0x800,idle_timeout=0,action=output:5
./utilities/ovs-ofctl add-flow br0 \
 in_port=6,dl_type=0x800,idle_timeout=0,action=output:2
./utilities/ovs-ofctl add-flow br0 \
```

```
 in_port=5,dl_type=0x800,idle_timeout=0,action=output:4

./utilities/ovs-ofctl dump-flows br0
```

12.3 Power on the VM

Start the first VM on CPU 4, CPU 5, and CPU 6 (0x70) with the following configuration:

```
# taskset 70 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/vm2-fc21.img -boot c -enable-kvm -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm2monitor,server,nowait -name 'FC21-VM2' -net none -no-reboot -object memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -numa node,memdev=mem -mem-prealloc  \
-chardev socket,id=char1,path=/usr/local/var/run/openvswitch/vhost-user0 \
-netdev type=vhost-user,id=net1,chardev=char1,vhostforce -device virtio-net-pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso6=off,guest_ecn=off,mrg_rxbuf=off \
-chardev socket,id=char2,path=/usr/local/var/run/openvswitch/vhost-user1 \
-netdev type=vhost-user,id=net2,chardev=char2,vhostforce -device virtio-net-pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso6=off,guest_ecn=off,mrg_rxbuf=off \
--nographic -vnc :14
```

12.3.1 VM Kernel Boot Parameters

1. Add the following to the kernel boot parameters /etc/default/grub in the VM:

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=fedora-server/root rd.lvm.lv=fedora-server/swap default_hugepagesz=1G hugepagesz=1G hugepages=1 hugepagesz=2M hugepages=1024 isolcpus=1,2 rhgb quiet"
```

2. Save the file and update GRUB config file:

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

3. Reboot the VM and then check to make sure 1GB and 2MB HugePage sizes are created. You should see one 1GB HugePages and 1024 2MB HugePages.

```
ls /sys/devices/system/node/node0/hugepages/hugepages-*

Output:
hugepages-1048576kB/    hugepages-2048kB/

cat /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
1

cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
1024
```

4. Start the second VM by making a copy of the first VM. Start the second VM on CPU 7, CPU8, and CPU9 (0x380) with the following command:

```
# taskset 380 qemu-system-x86_64 -m 4096 -smp 4 -cpu host -hda /root/vm-images/vm2-fc21.img -boot c -enable-kvm -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm2monitor,server,nowait -name 'FC21-VM2' -net none -no-reboot -object
```

```
memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -numa
node,memdev=mem -mem-prealloc  \
-chardev socket,id=char1,path=/usr/local/var/run/openvswitch/vhost-user0 \
-netdev type=vhost-user,id=net1,chardev=char1,vhostforce -device virtio-net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso6
=off,guest_ecn=off,mrg_rxbuf=off \
-chardev socket,id=char2,path=/usr/local/var/run/openvswitch/vhost-user1 \
-netdev type=vhost-user,id=net2,chardev=char2,vhostforce -device virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso6
=off,guest_ecn=off,mrg_rxbuf=off \
--nographic -vnc :15
```

12.4 Set up the VM HugePages

Mount the HugePage for 1GB and 2MB:

```
mount -t hugetlbfs hugetlbfs /mnt/huge
mount -t hugetlbfs none /mnt/huge_2mb -o pagesize=2MB
```

12.5 Set up DPDK 2.0

1. Download DPDK 2.0.0 and compile it:

```
make install T=x86_64-native-linuxapp-gcc
```

2. Edit the test-pmd apps input and output queue size to 2K for better throughput performance:

```
vi /root/dpdk-2.0.0/app/test-pmd/test-pmd.c

/*
* Configurable number of RX/TX ring descriptors.
*/

#define RTE_TEST_RX_DESC_DEFAULT 2048
#define RTE_TEST_TX_DESC_DEFAULT 2048
```

3. Save and build the test-pmd app:

```
export RTE_SDK=/root/dpdk-2.0.0
export RTE_TARGET=x86_64-native-linuxapp-gcc
make
```

12.6 Set up the vHost Network in the VM

1. Load the UIO kernel module in the VM:

```
modprobe uio
insmod /root/dpdk-2.0.0/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko
```

2. Check the PCI ID for the 10GbE NIC ports:

```
lscpi -nn
00:04.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
00:05.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
```

3. Bind the user side vhost network devices with igb_uio driver:

```
/root/dpdk-2.0.0/tools/dpdk_nic_bind.py -b igb_uio 00:04.0
/root/dpdk-2.0.0/tools/dpdk_nic_bind.py -b igb_uio 00:05.0
/root/dpdk-2.0.0/tools/dpdk_nic_bind.py --status

Network devices using DPDK-compatible driver
============================================
0000:00:04.0 'Virtio network device' drv=igb_uio unused=virtio_pci
0000:00:05.0 'Virtio network device' drv=igb_uio unused=virtio_pci

Network devices using kernel driver
===================================
<none>
```

12.7 Start test-pmd Application in the VM

1. Run test-pmd app on vCPU1 and vCPU2 (0x6):

```
cd /root/dpdk-2.0.0/x86_64-native-linuxapp-gcc/build/app/test-pmd
./testpmd -c 0x6 -n 4 -- --burst=32 -i --txd=2048 --rxd=2048 --txqflags=0xf00 --
disable-hw-vlan
```

2. In the application, enter the `fwd` and `mac_retry` commands:

```
testpmd> set fwd mac_retry
Set mac_retry packet forwarding mode
```

3. Start the PMD forwarding operation:

```
testpmd> start
mac_retry packet forwarding - CRC stripping disabled - packets/burst=32
nb forwarding cores=1 - nb forwarding ports=2
RX queues=1 - RX desc=2048 - RX free threshold=32
RX threshold registers: pthresh=8 hthresh=8 wthresh=0
TX queues=1 - TX desc=2048 - TX free threshold=0
TX threshold registers: pthresh=32 hthresh=0 wthresh=0
TX RS bit threshold=0 - TXQ flags=0xf00
```

12.8 CPU Affinity Tuning

The tables below show the host's CPU core affinity settings for VM-to-VM tests configuration for 1 physical core (no hyper-threading). When the two VMs start, there will be multiple QEMU threads spawned. Refer to section 9.4, CPU Core Affinity for Virtual Machine (qemu-kvm), to set the active QEMU threads to the correct core affinity.

CPU affinity setting on the host

Logical Core	Process
1	ovs-vswitchd
2	PMD0
4, 5, 6	QEMU (VM1)
7, 8, 9	QEMU (VM2)

QEMU threads CPU affinity

VM1 QEMU threads:

Logical Core	Process	CPU% (from htop)
4	QEMU (main thread for VM1)	100
5	QEMU	100
6	QEMU	0
6	QEMU	0
6	QEMU	0
6	QEMU	0

Note: Two active threads (with 100 CPU%) are set to 2 different logical cores

VM2 QEMU threads:

Logical Core	Process	CPU% (from htop)
7	QEMU (main thread for VM2)	100
8	QEMU	100
9	QEMU	0
9	QEMU	0
9	QEMU	0
9	QEMU	0

Note: Two active threads (with 100 CPU%) are set to 2 different logical cores

Acronyms and Abbreviations

Abbreviation	Description
BMWG	Benchmark Working Group
DUT	Device-Under-Test
EMC	Exact Match Cache
IETF	Internet Engineering Task Force
IPDV	Inter-Packet Delay Variation
KVM	Kernel-based Virtual Machine
NFV	Network Functions Virtualization
OPNFV	Open Platform for NFV
OVS	Open vSwitch
PDV	Packet Delay Variation
QEMU	Quick Emulator
SDN	Software-Defined Networking
VNF	Virtualized Network Function

Legal Information

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Intel technologies may require enabled hardware, specific software, or services activation. Check with your system manufacturer or retailer. Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

Intel does not control or audit third-party web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.