Intel® Open Network Platform Release 2.1 Performance Test Report

SDN/NFV Solutions with Intel® Open Network Platform

Document Revision 1.0
March 2016

Revision History

Date	Revision	Comments
March 31, 2016	1.0	Initial document for release of Intel® Open Network Platform Release 2.1

Contents

Figures

Tables

1.0 Audience and Purpose

Intel® Open Network Platform (Intel ONP) 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® ONP is released in the form of a software stack and a set of documents (e.g. Intel® Open Network Platform Reference Architecture Guide, Release Notes, and Performance Test Reports) available on 01.org.

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

- OpenStack*
- OpenDaylight*
- Data Plane Development Kit (DPDK)*
- Open vSwitch* with DPDK
- Fedora*.

This test report provides a guide to packet processing performance testing of the Intel® ONP. 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 and can greatly assist in achieving optimal system performance.

Ideally, the same hardware platform specifications and software versions are used for both Intel® ONP Reference Architecture Guide and Performance Test Reports. Exceptions can however occur due to software issues, version revisions and other factors during integration and benchmarking activities. Information on these exceptions is provided in *Intel® ONP Release 2.1 Application Note on Hardware and Software Differences between Reference Architecture Guide and Performance Test Report* available on 01.org.

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 essential when testing complex systems and this can be difficult to achieve with manual configuration and test methods. Automated integration, deployment and test methods are needed for developing robust SDN/NFV solutions. Many of these challenges are being addressed through industry forums such as OPNFV. Future versions of Intel® ONP will also provide guidance on adopting automation tools and methods.

This report is built on earlier Intel® ONP test reports (available on 01.org as archived documents). These earlier reports have baseline performance data and configuration procedures (for Linux operating system setup, BIOS configurations, configuration for OvS, VM setup, building DPDK and OvS, etc.).

The focus of this report is to present the packet processing performance using the Intel® Xeon® processor E5-2695 v4 (formerly Broadwell-EP) that can provide up to 18 Xeon®-class physical cores and is ideal for NFV solutions for hosting Virtual Network Functions (VNFs).

In this document "native OvS" refers to Open vSwitch which is a multilayer virtual switch licensed under the open source Apache 2.0 license (http://openvswitch.org/). The DPDK accelerated version of OvS is referred to as "OvS with DPDK" in this document.

This report includes:

- Packet performance tests on Intel® Xeon® processor E5-2695 v4 (formerly Broadwell-EP)
- New versions of software ingredients (compared to Intel® ONP2.0)
- vHost user for QEMU VM fast-path interface
- 40Gbps performance testing with two Intel® Ethernet X710-DA4 Adapters
- Updated Virtual eXtensible LAN (VXLAN) performance tests
- Tuning methods and troubleshooting tips for achieving good packet processing performance with Open vSwitch with DPDK
- Information on industry NFV test activities
- Performance using host with DPDK i.e. "bare-metal" (used to establish a performance baseline)
- Virtual Switch performance comparing native OvS and OvS with DPDK
- Performance scaling of OvS with DPDK
- Throughput performance with one and two VMs
- VXLAN performance comparing native OvS and OvS with DPDK.

3.0 Platform Specifications

3.1 Hardware Ingredients

Table 3-1 Intel® Xeon® processor E5-2695 v4 Platform – hardware ingredients used in performance tests

Item	Description
Server Platform	Supermicro X10DRH-I http://www.supermicro.com/products/motherboard/Xeon/C600/X10DRH-i.cfm Dual Integrated 1GbE ports via Intel® i350-AM2 Gigabit Ethernet
Chipset	Intel® C612 chipset (formerly Lynx-H Chipset)
Processor	1x Intel® Xeon® processor E5-2695 v4 (formerly Broadwell-EP) 2.10 GHz; 120 W; 45 MB cache per processor 18 cores, 36 hyper-threaded cores per processor
Memory	64GB Total; Samsung 8GB 2Rx8 PC4-2400MHz, 8GB per channel, 8 Channels
Local Storage	500 GB HDD Seagate SATA Barracuda 7200.12 (SN:Z6EM258D)
PCIe	2 x PCI-E 3.0 x8 slot
NICs	2 x Intel® Ethernet Converged Network Adapter X710-DA4 Total: 8 Ports; 2 ports from each NIC used in tests.
BIOS	AMIBIOS Version: 2.0 Release Date: 12/17/2015

3.2 Software Versions

Table 3-2 Software versions used in performance tests

Software Component	Version
Host Operating System	Fedora 23 x86_64 (Server version) Kernel version: 4.2.3-300.fc23.x86_64
VM Operating System	Fedora 23 x86_64 (Server version) Kernel version: 4.2.3-300.fc23.x86_64
QEMU-KVM	QEMU-KVM version 2.5.0 libvirt version: 1.2.18.2-2.fc23.x86_64
Open vSwitch (native OvS and OvS with DPDK)	Open vSwitch 2.4.9 Commit ID: 53902038abe62c45ff46d7de9dcec30c3d1d861e
Intel® Ethernet Drivers	i40e-1.4.25 • Intel® Ethernet Converged Network Adapters X710-DA4
DPDK	DPDK version: 2.2.0 http://www.dpdk.org/browse/dpdk/snapshot/dpdk-2.2.0.tar.gz

3.3 Boot Settings

Although Turbo Boost and Energy Efficient Turbo would increase the performance, not all cores would be running at the turbo frequency. CPU turbo frequency would change depending on the number of active cores being utilized and it would impact the consistency of the OvS throughput performance. Thus, it would not be a real representation of the platform performance. Therefore, test configuration in Intel® ONP 2.1 was changed respectively to have the Turbo Boost, Energy Efficient Turbo and P-state disabled in BIOS.

Table 3-3 Boot Settings

System Capability	Description
Host Boot Settings	Hugepage size = 1G ; No. of Hugepages = 16 Hugepage size=2MB; No. of Hugepages = 2048 Hyper-threading disabled: isolcpus = 1-17 Hyper-threading enabled: isolcpus = 1-17,19-35
BIOS settings	C-state disabled P-state disabled Hyper-threading enabled/disabled Turbo Boost Disabled Enhanced Intel® Speedstep® Technology disabled Energy Efficient Turbo disabled Intel VT-x enabled
VM Kernel Boot settings	Hugepage size = 1G ; No. of Hugepages = 1 Hugepage size=2MB; No. of Hugepages = 1024 isolcpus = 1-2

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 CLFAGS="-Ofast -g"
DPDK Settings	Build L3fwd: (in l3fwd/main.c) #define RTE_TEST_RX_DESC_DEFAULT 2048 #define RTE_TEST_TX_DESC_DEFAULT 2048

3.5 Operating System Settings

Table 3-5 Operating System Settings

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 # kill -9 dhclient # echo 0 > /proc/sys/kernel/randomize_va_space SELinux disabled net.ipv4.ip_forward=0
Linux Module Settings	# rmmod ipmi_msghandler # rmmod ipmi_si # rmmod ipmi_devintf # rmmod lpc_ich # rmmod bridge

4.0 Test Configurations

The test setup is shown in Figure 4-1. The system-under-test is Intel® ONP Reference Architecture Release 2.1. The traffic is generated by Ixia running RFC 2544 (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 40 Gbps aggregated across four 10GbE ports, except for VXLAN tests which use two ports. Physical ports are paired (one as ingress and one as egress), i.e., one 10 Gbps bidirectional flow "consumes" two ports. Unless otherwise stated, all tests are for zero packet loss.

Note: The System has two 4x10 Gbe formerly Fortville NICs. Two ports from each NICs are used in the setup to achieve the maximum packets performance in the system under test.

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

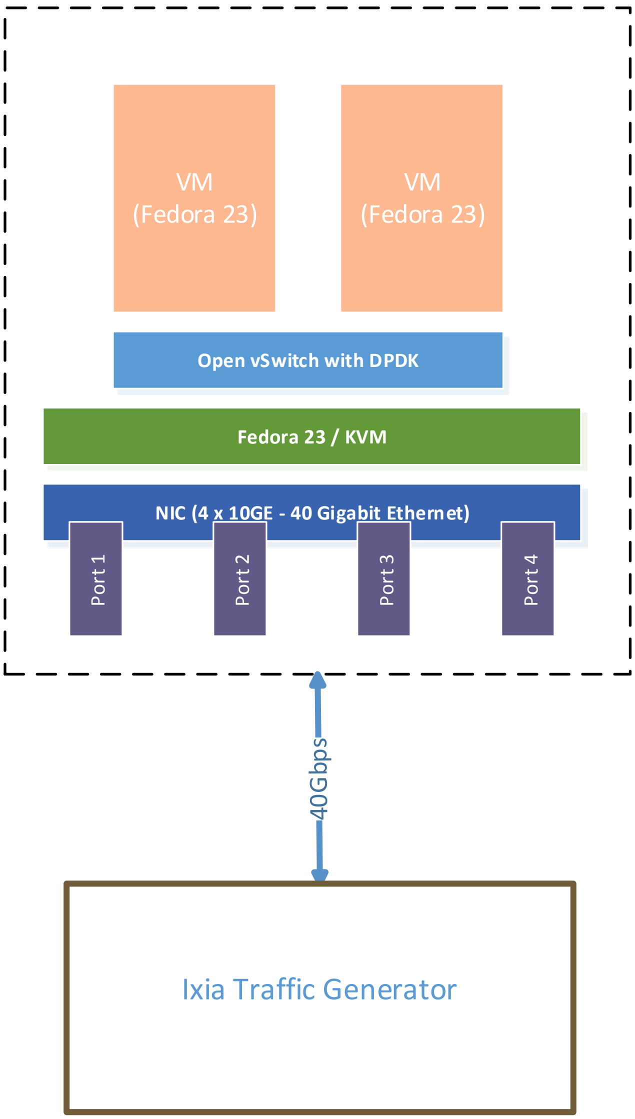

Figure 4-1 High-Level Overview of Test Setup

Allocation of cores has large impact on performance. This document include test configurations with hyper-threading enabled and disabled as well as 1, 2 and 4 physical cores. Table 4-1 summarizes the combinations of physical and hyper-threaded cores used for each test case.

Table 4-1 Number of cores used for each test category (4 physical cores total per CPU)

Test	Hyper-Threading off Number of Physical cores			Hyper-Threading on Number of Hyper-Threaded Cores	
	1	2	4	2	4
Host Tests	✓	✓	✓		
Virtual Switching Tests	✓	✓	✓	✓	✓
PHY-to-VM Tests	✓	✓	✓	✓	✓
VM-to-VM Tests	✓	✓	✓	✓	✓
VXLAN Tests	✓	✓			

4.1 Traffic Profiles

The IP traffic profile used conforms to RFC 2544:

- Frame sizes (bytes): 64, 128, 256, 512, 1024, 1280, and 1518
- L3 protocol: IPv4
- L4 protocol: UDP
- All tests are bidirectional with the same data rate being offered from each direction.
- For VXLAN, a header is used to encapsulate IP packets per RFC 7348.

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 specifies the following types of tests:

- **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 service-level agreements (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 a particular packet size with an offered load and assumes the system has reached a steady state. Using the RFC 2544 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 illustrated in Table 5-1 using two examples of configurations with two and three switching operations, respectively. Table 5-1 shows the two configuration examples. Careful analysis of all configuration variables is needed before making performance comparisons that are meaningful.

Table 5-1 Throughput and switching performance metrics for two example test-cases

Parameter	Configuration Examples	
	PHY-OVS-PHY	PHY-VM-VM-PHY
Physical Ports	4	2
Physical cores	4	4
Hyper-threaded cores	4	4
Flows per Port (in each direction)	1	1
Total Flows	4	2
Switching Operations	1	3
Line-rate	40 Gbps	20 Gbps
OvS with DPDK: Throughput (packets/sec) 128B packets	33,783,782	5,345,806
OvS with DPDK: Switching (packets/sec) 128B packets	33,783,782	16,037,417

In this example, while the number of physical cores and hyper-threaded cores are same in both scenarios, the number of switching operations is different. Line rate is therefore different and cannot be used as a comparison metric. A more reasonable comparison would be to compare the switching performance as measured by the throughput (packets/second) multiplied by the number of switching operations. In this case we can see that adding 2 VMs introduces two extra switching operations and results in approximately 52% degradation in performance compared to the case with only a vSwitch and no VMs (single switching operation).

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 RFC 2544 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 Telecommunication Standardization Sector 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 measuring variation of packet delay:

- 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 RFC 3550.
- 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 [RFC 3393] and periodic streams [RFC 3432], 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
Host Performance (PHY-PHY)					
7.1	L3 Fwd (no pkt modification) 4 ports	Throughput Latency (min, max, avg)	64, 72, 128, 256, 512, 768, 1024, 1280, 1518	60 sec	One flow/port in both directions
vSwitch Performance (PHY-OVS-PHY)					
7.2	L3 Fwd 4 ports	Throughput Latency (min, max, avg)	64, 72, 128, 256, 512, 768, 1024, 1280, 1518	60 sec	One flow/port in both directions; One thousand flows/port in both directions
One VM Throughput (PHY-VM-PHY)					
7.3	Single VM (vhost-user) L3 Fwd 4 ports	Throughput Latency (avg)	64, 256	60 sec	One flow/port in both directions; One thousand flows/port in both directions
Two VM Throughput (PHY-VM-VM-PHY)					
7.4	Two VMs in series (vhost-user) L3 Fwd 2 ports	Throughput Latency (min, max, avg)	64, 72, 128, 256, 512, 768, 1024, 1280, 1518	60 sec	One flow/port in both directions
40 Gbps switching Performance					
7.5	4 VMs (4 x PHY-VM) 4 VMs (2 x PHY-VM and 1 x PHY-VM-VM-PHY) 2 VMs in series (PHY-VM-VM-PHY) 2ports	Throughput	256	60 sec	One flow/port in both directions
VXLAN Performance (PHY-OVS-VM-OVS-PHY)					
7.6	VXLAN decap/encap using vSwitch Tunnel End Point (TEP) with 1 VM L3 Fwd 1 port	Throughput Latency (min, max, avg)	114, 122, 178, 306, 562, 818, 1074, 1330, 1468	60 sec	One flow/port

7.0 Test Results

7.1 Host Performance (PHY-PHY)

The test setup for measuring host (i.e. no VM) throughput performance is shown in Figure 7-1. The host is installed with DPDK and it uses DPDK's L3 forwarding sample application. This test creates a good baseline for comparing more complex test cases as well as comparing "bare-metal" performance between different platforms. This is very important when trying to establish "apples-for-apples" comparisons for more complex test scenarios between different platforms. If the "bare-metal" performance cannot be calibrated between the platforms on a per-core / per-port basis, more complex scenarios involving virtualization will certainly not provide valid comparisons. Host tests attempt to achieve system throughput of 40 Gbps, using a 4-port configuration with 4 physical cores.

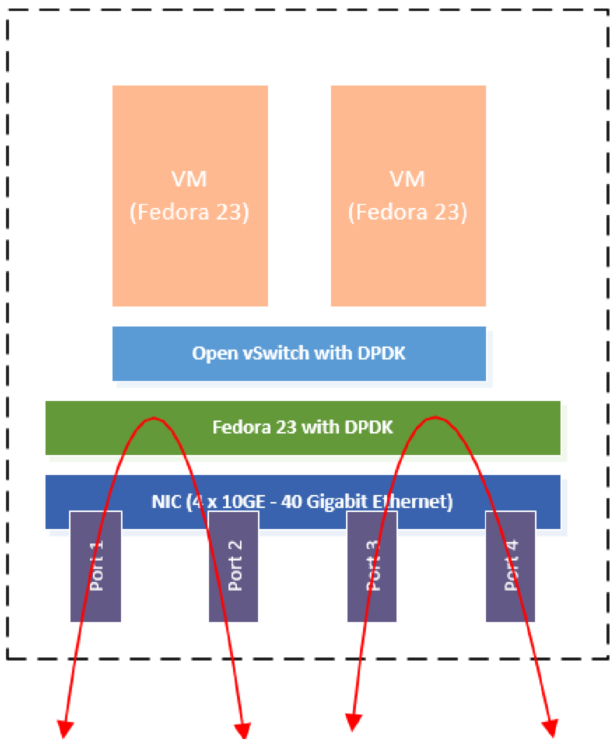

Figure 7-1 Host Performance test setup (PHY-PHY)

Table 7-1 summarizes the permutations of test configuration variables. In this case all tests uses 4 ports, 1 TX/RX queue per port, and 1 bi-directional flow per port (i.e. total of 4 unidirectional flows). Permutations were tested with 1, 2, and 4 physical cores and hyper-threading was not enabled.

Table 7-1 PHY-PHY test configuration variables

	Configuration Variable				
Test	Ports	TX/RX Queues per Core	Flows per Port in Each Direction	Physical Cores	Hyper-Threaded Cores
L3 Forwarding	4	1	1	1, 2, 4	0

Test results are shown in the Figure 7-2 below for all packet sizes.

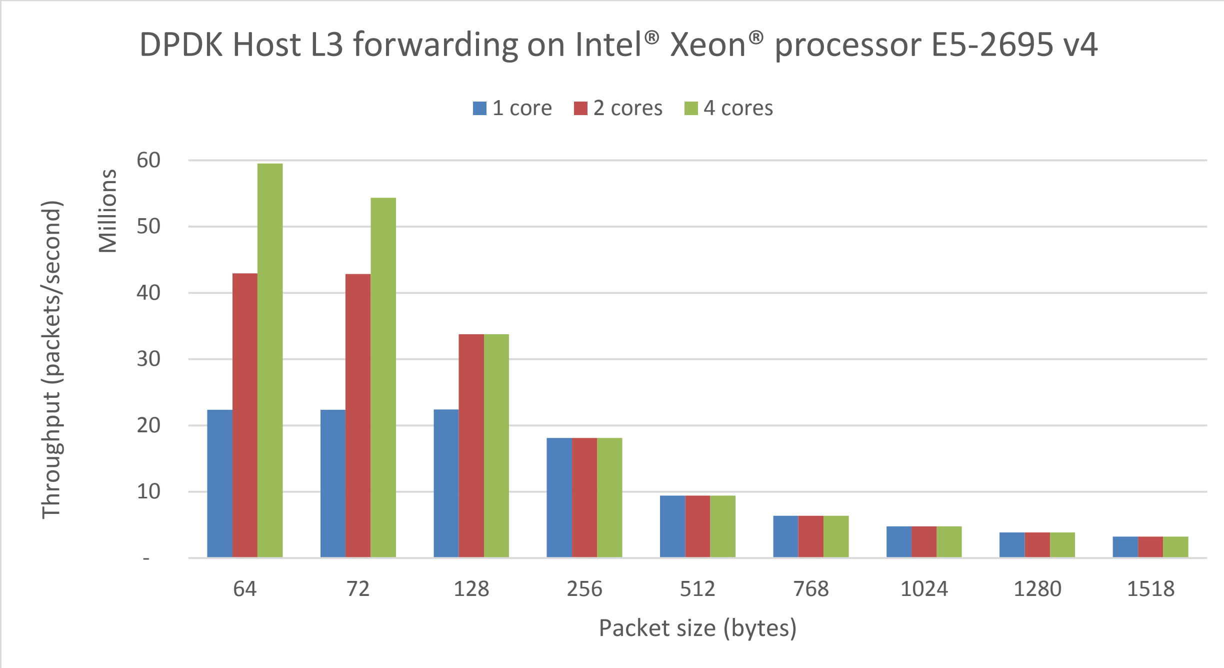

Figure 7-2 Forwarding throughput performance with 1, 2, and 4 physical cores

L3 test data is presented for the **2 cores** configuration in the Table 7-2. Line-rate (i.e. 40 Gbps) is achieved for all packet sizes equal or larger than 128 bytes.

Table 7-2 PHY-PHY test results for 2 physical cores

L3 Forwarding — Bidirectional Throughput with Zero Packet Loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Min Latency (ns)	Average Latency (ns)	Maximum Latency (ns)
64	28,863	42,950,430	72	6,820	87,828	201,900
72	31,531	42,840,797	79	7,060	90,300	210,860
128	40,000	33,783,767	100	12,180	21,468	84,740
256	40,000	18,115,934	100	14,180	18,767	59,820
512	40,000	9,398,494	100	18,080	23,722	41,220
768	40,000	6,345,173	100	24,700	29,255	89,620
1024	40,000	4,789,269	100	31,160	35,450	79,100
1280	40,000	3,846,151	100	16,220	41,872	108,180
1518	40,000	3,250,972	100	31,380	47,686	109,240
Affinity Summary	1 TxRx queue per core 2 logical core on 2 physical core (SMT not used)					
Affinity Details	0.0% Loss resolution ./l3fwd -c 0xc -n 4 -- -p 0xf --config="(0,0,2)(1,0,3)(2,0,2)(3,0,3)" 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC					

7.2 Virtual Switching Performance (PHY-OVS-PHY)

Figure 7-3 shows test setup for PHY-OVS-PHY with four 10GbE ports.

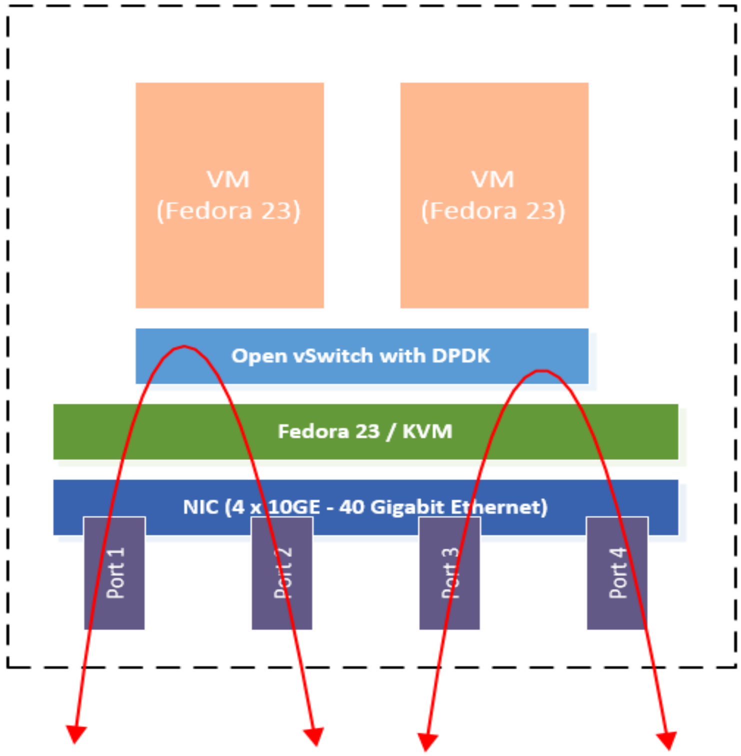

Figure 7-3 Virtual Switching Performance test setup (PHY-OVS-PHY)

Virtual switching tests attempt to achieve aggregated system throughput of 40 Gbps using 4 ports to compare the following variables' configurations:

- Native OvS versus OvS with DPDK
- 1, 2, or 4 physical cores
- One flow per port (total four flows) or 1K flows per port (total 4K flows)
- Hyper-threading on or off
- 1 physical core vs 2 hyper-threaded cores
- 2 physical cores vs 4 hyper-threaded cores.

7.2.1 Native OvS and OvS with DPDK

This test compares the throughput performance of native OvS and OvS with DPDK for various packet sizes, assuming one flow per port and single physical core in use (hyper-threads enabled and disabled). The data shows that for small packet sizes throughput increases by over 10x with hyper-threading disabled and 11.7x with hyper-threading enabled using OvS with DPDK. However, with hyper-threading enabled, average latency is increased by 2 times for 64B packets comparing to results without hyper-threading, because the hyper-threaded cores are contending for cache-line access and causes thrashing to occur.

Table 7-3 PHY-OVS-PHY test configuration variables for native OvS and OvS with DPDK

Configuration variables	Ports	Flows per Port in each Direction	Physical Cores	Hyper-threaded cores
Native OvS	4	1	1	2
OvS with DPDK	4	1	1	0, 2

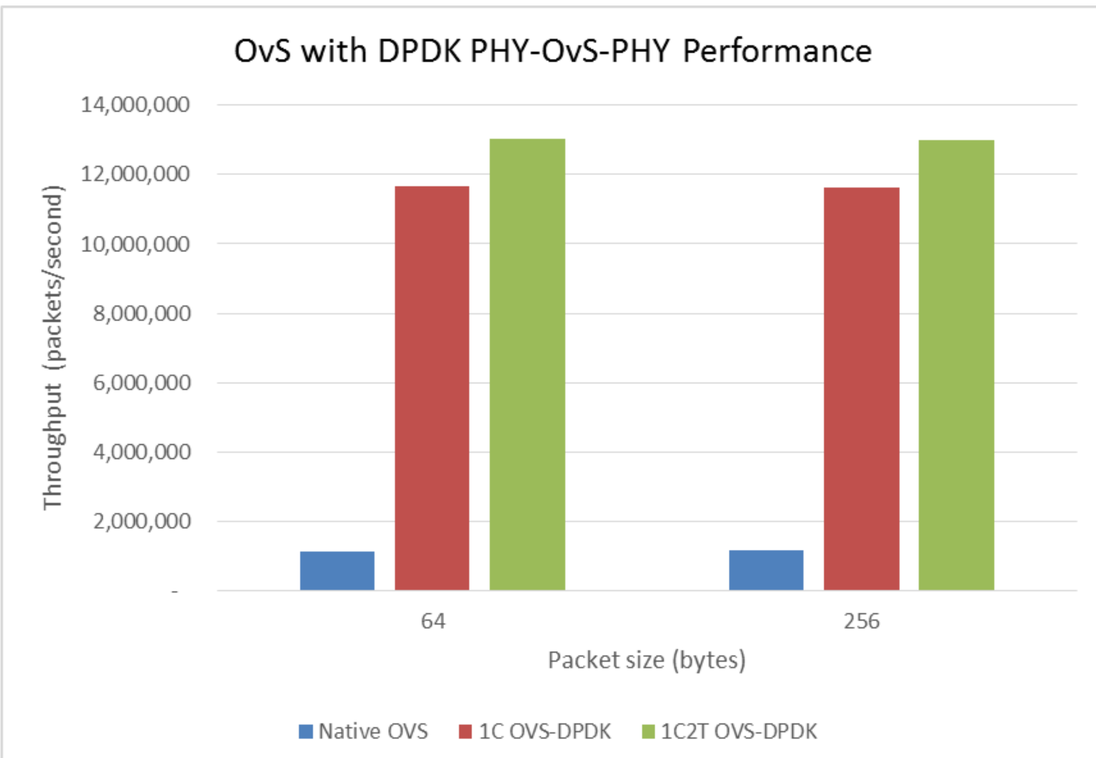

Figure 7-4 Throughput performance of native OvS (with hyper-threading) and OvS with DPDK (without and with hyper-threading)

Table 7-4 PHY-OVS-PHY test results for native OvS (1 core, 1 flow per port)

L3 Forwarding — Bidirectional Throughput with Zero Packet Loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (ns)	Minimum Latency (ns)	Maximum Latency (ns)
64	748	1,113,190	2	57,913	5,940	8,463,600
72	864	1,174,039	2	56,699	5,860	13,334,820
128	1,483	1,252,418	4	55,136	4,540	9,330,820
256	2,566	1,162,006	6	57,058	6,160	8,477,940
512	4,422	1,038,999	11	59,589	6,340	5,849,020
768	6,278	995,915	16	60,711	6,340	4,966,380
1024	7,709	923,028	19	62,158	6,840	4,736,480
1280	9,604	923,472	24	63,741	7,320	5,405,040
1518	9,952	808,857	25	67,298	8,640	4,951,300
Affinity Summary	1 TxRx queue per core 1 logical core on 1 physical core (SMT not used)					
Affinity Details	0% Loss resolution 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC Port0 IRQ's Affinity to lcore2 Port1 IRQ's Affinity to lcore3 Port2 IRQ's Affinity to lcore4 Port3 IRQ's Affinity to lcore5					

Table 7-5 PHY-OVS-PHY test results for OvS with DPDK (1 core, 1 flow per port)

L3 Forwarding — Bidirectional Throughput with Zero Packet Loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (ns)	Minimum Latency (ns)	Maximum Latency (ns)
64	7,825	11,644,358	20	25,486	10,200	123,240
72	8,560	11,630,128	21	19,968	8,820	76,700
128	13,780	11,638,931	34	27,153	12,560	84,600
256	25,653	11,618,095	64	21,489	13,440	78,340
512	40,000	9,398,496	100	37,271	16,800	57,840
768	40,000	6,345,178	100	28,412	13,320	55,000
1024	40,000	4,789,272	100	31,387	11,180	56,120
1280	40,000	3,846,154	100	25,843	10,260	51,420
1518	40,000	3,250,975	100	27,232	10,080	48,580
Affinity Summary	1 TxRx queue per core 1 logical core on 1 physical core (SMT not used)					
Affinity Details	1PMD thread based OVS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=4 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC					

Table 7-6 PHY-OVS-PHY test results for OvS with DPDK (1 core with HT, 1 flow per port)

L3 Forwarding — Bidirectional Throughput with Zero Packet Loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (ns)	Minimum Latency (ns)	Maximum Latency (ns)
64	8,753	13,025,495	22	50,750	8,020	583,980
72	9,565	12,996,245	24	19,803	8,040	137,160
128	15,405	13,010,743	39	31,497	8,720	265,980
256	28,708	13,001,755	72	30,777	11,680	277,260
512	40,000	9,398,497	100	28,071	16,620	46,240
768	40,000	6,345,179	100	25,217	12,940	48,940
1024	40,000	4,789,273	100	23,473	11,760	42,720
1280	40,000	3,846,154	100	21,684	11,300	39,680
1518	40,000	3,250,976	100	20,290	10,940	39,280
Affinity Summary	1 TxRx queue per core 2 logical core on 1 physical core (SMT used)					
Affinity Details	2PMD thread based OVS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=100004 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC					

7.2.2 Performance Scaling of OvS with DPDK

Figure 7-5 shows throughput performance of processing 64-byte packets with 1, 2, and 4 physical cores and compares 4 flows (1 flow per port) with 4K total flows (1K flows per port). This data show fairly linear performance scaling as the number of cores is increased.

Today, 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, a larger numbers of flows may result in packets using a slow data path.

Table 7-7 PHY-OVS-PHY test configuration variables for OvS with DPDK (scaling with physical cores)

Configuration variables	Ports	Flows per Port in each Direction	Physical Cores	Hyper-threaded cores
OvS with DPDK	4	1, 1k	1, 2, 4	0

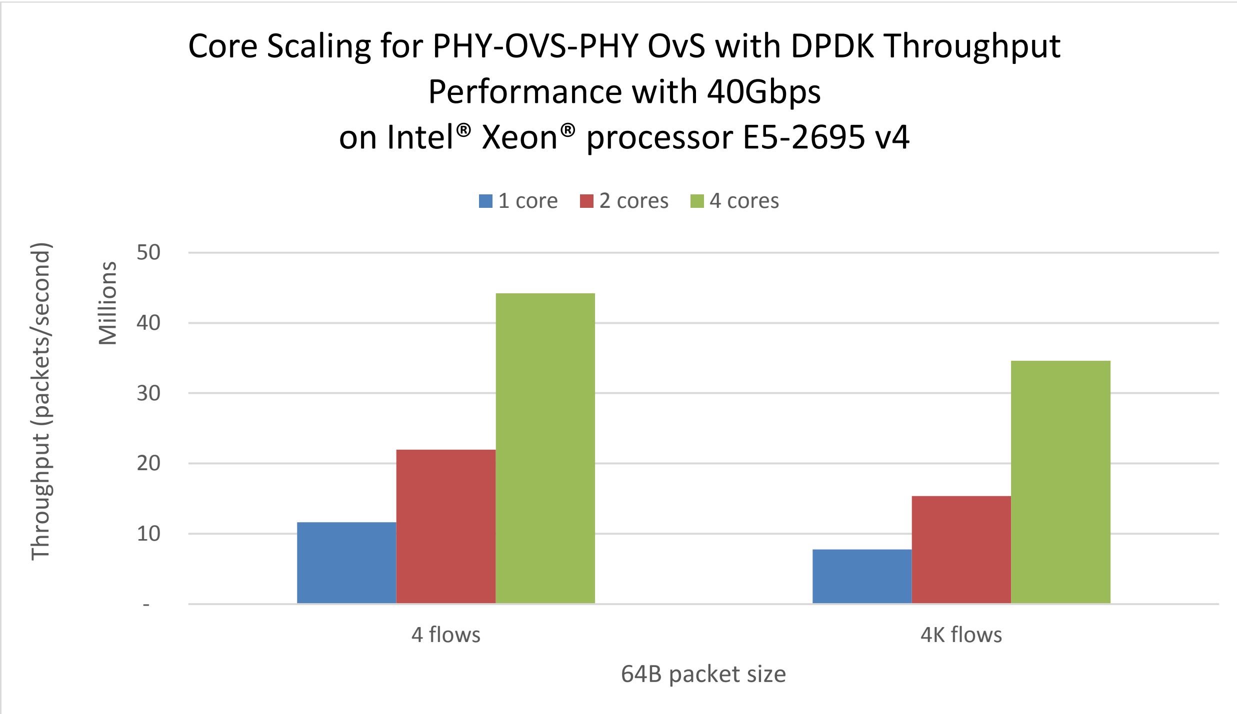

Figure 7-5 64-byte throughput performance scaling of OvS with DPDK for 1, 2, 4 cores, no hyper-threading

Table 7-8 PHY-OVS-PHY test results for OvS with DPDK (4 cores, 1k flows per port, HT off)

L3 Forwarding — Bidirectional Throughput with Zero Packet Loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (ns)	Minimum Latency (ns)	Maximum Latency (ns)
64	23,255	34,605,824	58	10,868	7,380	131,320
72	25,885	35,169,672	65	12,883	8,160	157,540
128	40,000	33,783,761	100	65,189	14,180	118,380
256	40,000	18,115,943	100	17,872	9,920	37,420
512	40,000	9,398,497	100	16,555	8,640	34,860
768	40,000	6,345,178	100	17,696	8,380	34,260
1024	40,000	4,789,272	100	18,591	8,200	36,340
1280	40,000	3,846,154	100	18,183	7,980	40,900
1518	40,000	3,250,975	100	19,880	7,680	37,300
Affinity Details	4PMD thread based OvS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=3c 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC					

7.2.3 Performance with Hyper-Threading

The impact of hyper-treading on performance is illustrated below by comparing 64B throughput with 1 and 2 physical cores and hyper-threading turned on and off. There is 12% performance gain with 1 core and hyper-threading enabled, while 19% performance gain with 2 cores and hyper-threading enabled at 64B packets. With hyper-threading enabled, we gain the advantage of the throughput performance increment, but there is trade off in latency performance at 64B packets, with the observation of higher latency shown in section 7.2.1.

Table 7-9 PHY-OVS-PHY test configuration variables for OvS with DPDK and 64-byte packets (impact of HT)

Configuration variables	Ports	Flows per Port in each Direction	Physical Cores	Hyper-threaded cores
OvS with DPDK	4	1, 1k	1, 2	0, 2, 4

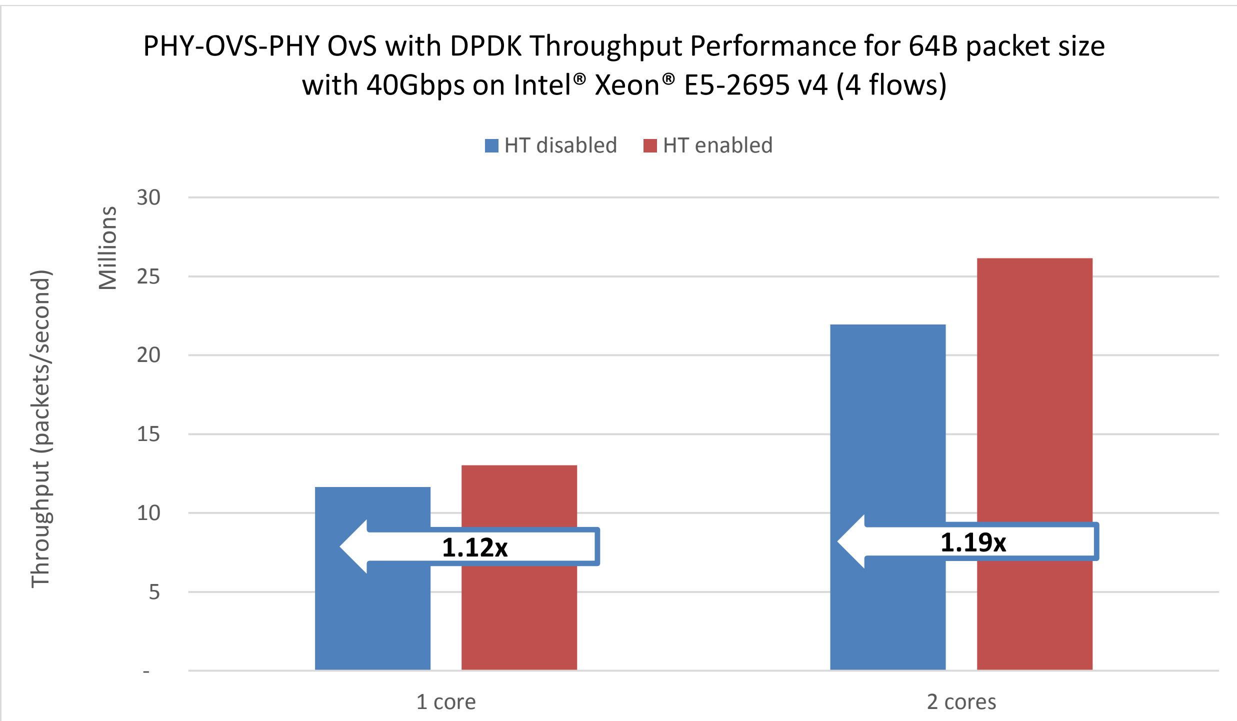

Figure 7-6 Impact of Hyper-threading on throughput performance of OvS with DPDK (64-byte packets, 1 and 2 cores, 4 flows)

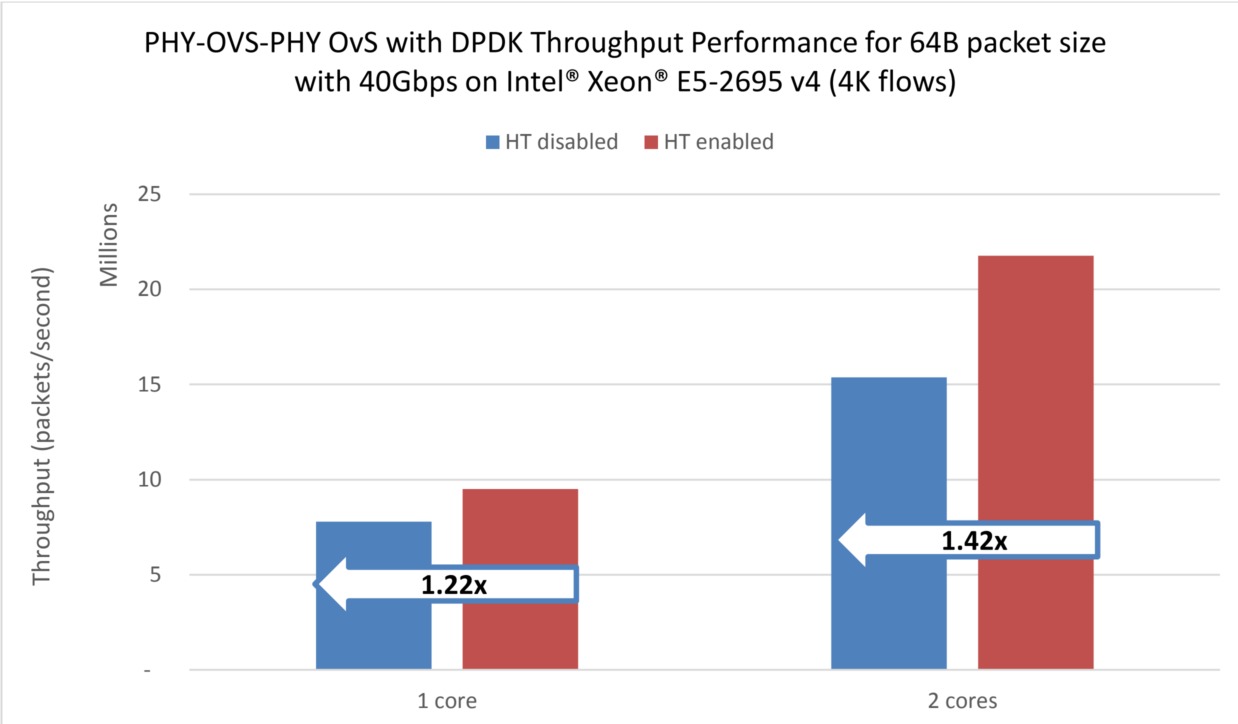

Figure 7-7 Impact of Hyper-threading on throughput performance of OvS with DPDK (64-byte packets, 1 and 2 cores, 4k flows)

7.3 One VM Throughput (PHY-VM-PHY)

This test uses a single VM with two bidirectional flows across four 10 GbE ports as shown in Figure 7-8. Maximum theoretical platform throughput is 40Gbps (four flows aggregated).

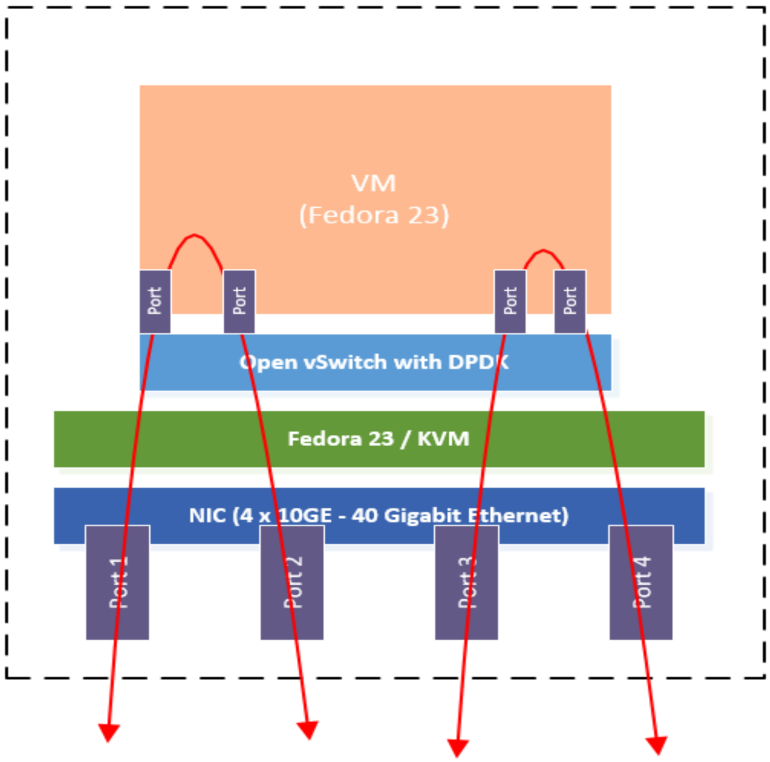

Figure 7-8 One VM Throughput Performance test setup

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

The single VM tests attempt to achieve aggregated system throughput of 40 Gbps using 4 ports to compare the following configurations for small packet sizes:

- L3 performance of native OvS versus OvS with DPDK
- L3 performance of OvS with DPDK
- Hyper-threading on or off (1 flow per port and 1k flows per port using 1 or 2 physical cores)
- Using 1 core, 2 cores, 4 cores (1 flow per port and 1k flows per port).

7.3.1 Native OvS and OvS with DPDK

This test compares the throughput performance of native OvS and OvS with DPDK.

In Figure 7-9 both native OvS and OvS with DPDK use one physical core. The relative performance ratio of OvS with DPDK is shown with and without hyper-threading.

Table 7-10 PHY-VM-PHY test configuration variables for native OvS and OvS with DPDK

Configuration variables	Ports	Flows per Port in each Direction	Physical Cores	Hyper-threaded cores
Native OvS	4	1	1	0
OvS with DPDK	4	1	1	0, 2

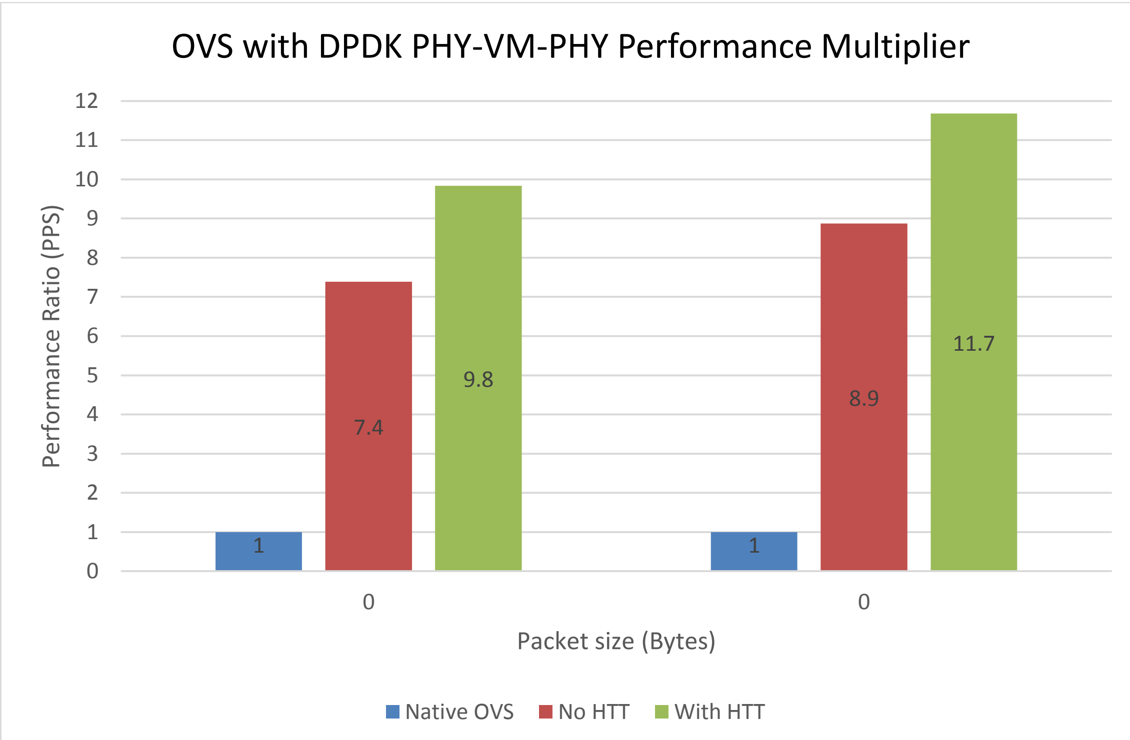

Figure 7-9 Throughput performance with 1 VM comparing native OvS and OvS with DPDK (using one physical core)

Table 7-11 shows measured small packet performance data for native OvS with hyper-threading on. Throughput performance is ~1% of line-rate at 64B packets. Table 7-12 shows measured small packet performance of OvS with DPDK also with one physical core (with hyper-threading). The throughput performance of OvS with DPDK is 7.4 times and 9.8 times greater than native OvS for 64B packets without and with hyper-threading respectively. There is a reduction of average latency at about 33% and 58% by using DPDK with OvS for 1 core configuration without and with hyper-threading respectively compared to native OvS at 64B packets.

Table 7-11 PHY-VM-PHY test results for native OvS (1 core, 1 flow per port)

Packet Size	L3 Forwarding — Bidirectional			
	Throughput with Zero Packet Loss			
	Mbps	Packets/sec	% Line Rate	Average Latency (ns)
64	316	470,262	1	141,000
256	799	362,054	2	113,768
Affinity Details	0% Loss resolution 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC Port0 IRQ's Affinity to lcore2 Port1 IRQ's Affinity to lcore3 Port2 IRQ's Affinity to lcore4 Port3 IRQ's Affinity to lcore5 On a VM: ./testpmd -c 0x6 -n 4 -- --burst=64 -i --txd=2048 --rxd=2048 --txqflags=0xf00			

Table 7-12 PHY-VM-PHY test results for OvS with DPDK (1 core, 1 flow per port)

Packet Size	L3 Forwarding — Bidirectional			
	Throughput with Zero Packet Loss			
	Mbps	Packets/sec	% Line Rate	Average Latency (ns)
64	2,334	3,472,608	6	94,077
256	7,090	3,211,156	18	84,118
Affinity Details	1PMD thread based OvS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=4 On a VM: ./testpmd -c 0x6 -n 4 -- --burst=64 -i --txd=2048 --rxd=2048 --txqflags=0xf00 --disable-hw-vlan 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC			

Table 7-13 PHY-VM-PHY test results for OvS with DPDK (1 core with HT, 1 flow per port)

Packet Size	L3 Forwarding — Bidirectional			
	Throughput with Zero Packet Loss			
	Mbps	Packets/sec	% Line Rate	Average Latency (ns)
64	3,107	4,623,558	8	59,170
256	9,333	4,226,995	18	56,986
Affinity Details	2PMD thread based OvS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=400004 On a VM: ./testpmd -c 0x6 -n 4 -- --burst=64 -i --txd=2048 --rxd=2048 --txqflags=0xf00 --disable-hw-vlan 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC			

Figure 7-10 shows improved performance of OvS with DPDK using two physical cores instead of only one. This test illustrates the ability of OvS with DPDK to increase performance by using additional cores. For both 64B and 256B packets the performance with two cores is nearly twice as high as with one core.

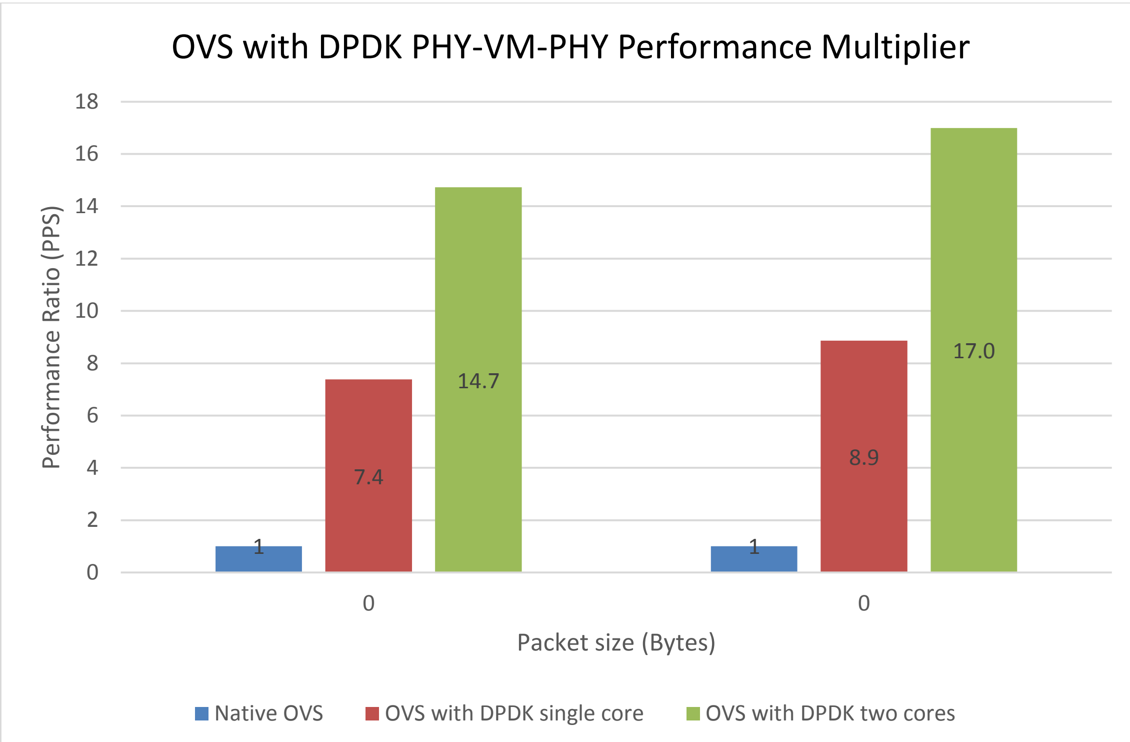

Figure 7-10 Throughput performance with a 1 VM comparing native OvS and OvS with DPDK using one and two physical cores (no hyper-threading)

Table 7-14 shows measured small packet performance data for OvS with DPDK using 2 physical cores.

Table 7-14 OvS with DPDK, no hyper-threading (2 physical cores)

	L3 Forwarding — Bidirectional			
	Throughput with Zero Packet Loss			
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (ns)
64	4,654	6,925,459	12	45,013
256	4,847	6,585,960	34	57,020
Affinity Details	2PMD thread based OvS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=c On a VM: ./testpmd -c 0x6 -n 4 -- --burst=64 -i --txd=2048 --rxd=2048 --txqflags=0xf00 --disable-hw-vlan 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC			

7.3.2 Performance Scaling of OvS with DPDK

Table 7-15 PHY-VM-PHY test configuration variables for OvS with DPDK (scaling with physical cores)

Configuration variables	Ports	Flows per Port in each Direction	Physical Cores	Hyper-threaded cores
OvS with DPDK	4	1, 1k	1, 2, 4	0

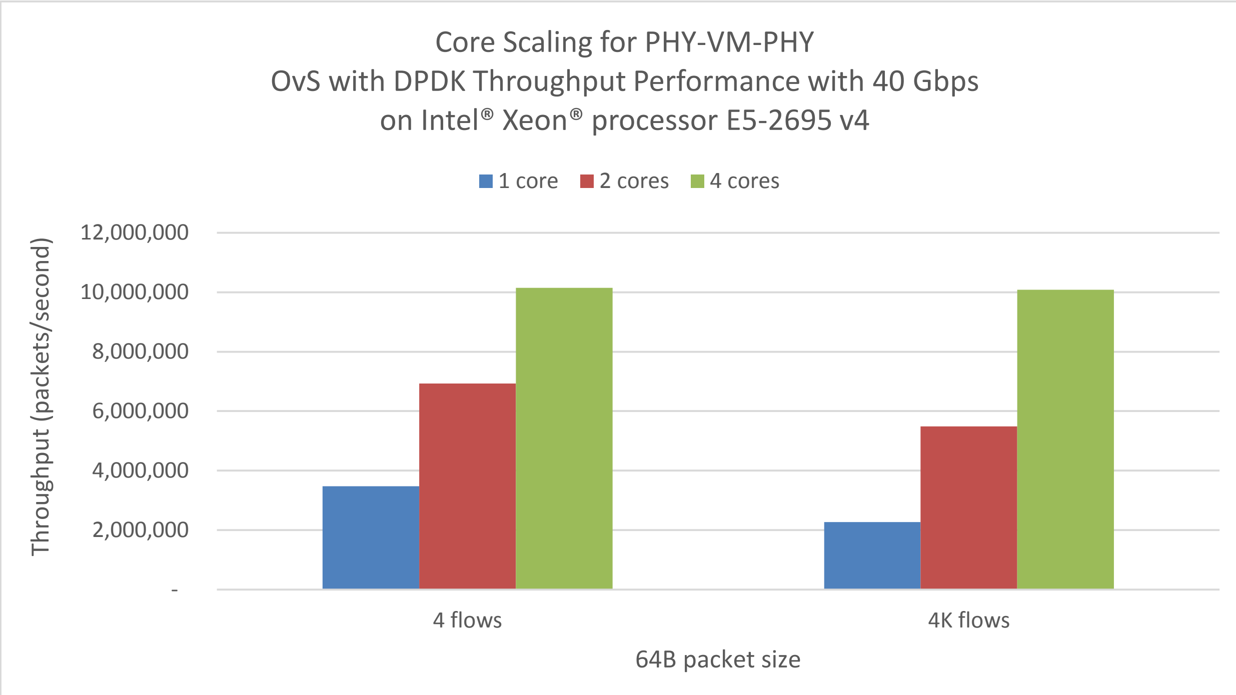

Figure 7-11 64-byte throughput performance scaling of OvS with DPDK - 1 VM (PHY-VM-PHY) with 1, 2, 4 cores, no hyper-threading

Table 7-16 PHY-VM-PHY test results for OvS with DPDK (1, 2, 4 cores, 1 flow per port, HT off)

	L3 Forwarding — Bidirectional			
	64B throughput with Zero Packet Loss			
Number of cores	Mbps	Packets/sec	% Line Rate	Average Latency (ns)
1	2,334	3,472,608	6	94,077
2	4,654	6,925,459	12	45,013
4	6,820	10,148,118	17	28,135
Affinity Details	1PMD, 2PMD, 4PMD thread based OvS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=4 (1 core) ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=c (2 cores) ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=3c (4 cores) On a VM: ./testpmd -c 0x6 -n 4 -- --burst=64 -i --txd=2048 --rxd=2048 --txqflags=0xf00 --disable-hw-vlan 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC			

7.3.3 Performance with Hyper-Threading

The test data in this section compares throughput performance with 1 VM using DPDK with and without hyper-threading (1 flow per port and 1k flows per port using 1 or 2 physical cores). There is about 33% and 61% gain in performance throughput for 1 core with hyper-threading enabled with 4 flows and 4k flows respectively. For 2 cores configuration, there is about 31% and 44% gain in performance throughput with hyper-threading enabled with 4 flows and 4k flows respectively. See Figure 7-12 and Figure 7-13.

Table 7-17 PHY-VM-PHY test configuration variables for OvS with DPDK and 64-byte packets (impact of HT)

Configuration variables	Ports	Flows per Port in each Direction	Physical Cores	Hyper-threaded cores
OvS with DPDK	4	1, 1k	1, 2	0, 2, 4

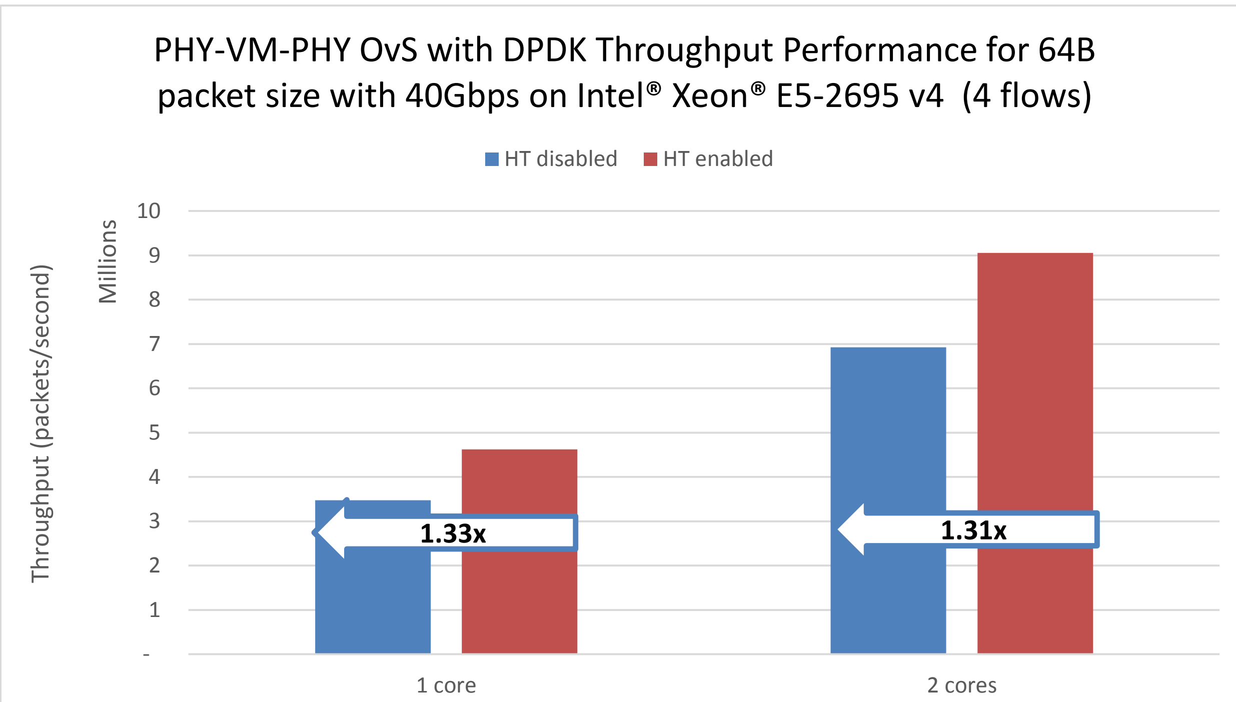

Figure 7-12 Impact of Hyper-threading on throughput performance of OvS with DPDK (64-byte packets, 1 and 2 cores, 4 flows, 1 VM)

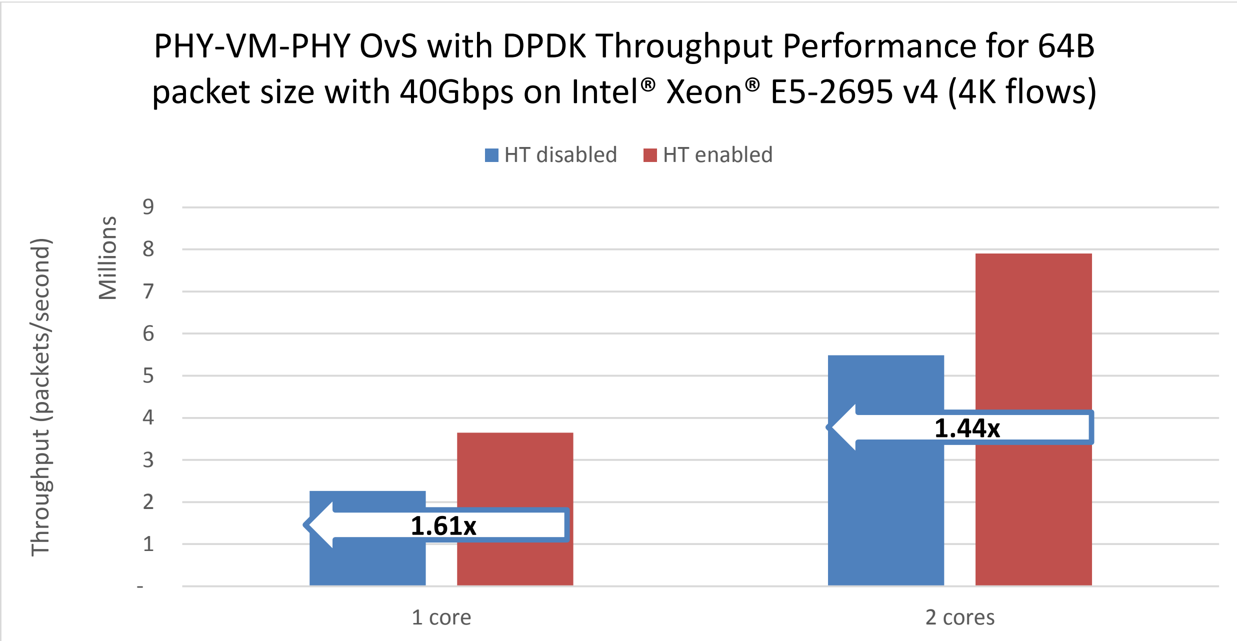

Figure 7-13 Impact of Hyper-threading on throughput performance of OvS with DPDK (64-byte packets, 1 and 2 cores, 4k flows, 1 VM)

Table 7-18 shows the measured performance data for 2 physical cores with hyper-threading off and on.

Table 7-18 PHY-VM-PHY test results for OvS DPDK (2 physical cores)

		L3 Forwarding — Bidirectional			
		64B throughput with Zero Packet Loss			Average Latency (ns)
	Hyper-threading	Mbps	Packets/sec	% Line Rate	
1 flow per port	Off	4,654	6,925,459	12	45,013
	On	6,085	9,054,718	15	38,634
1k flows per port	Off	3,687	5,486,768	9	100,815
	On	5,311	7,903,768	13	65,898
Affinity Details		2PMD and 4PMD thread based OvS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=C (HT off) ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask= 30000C (HT on) On a VM: ./testpmd -c 0x6 -n 4 -- --burst=64 -i --txd=2048 --rxd=2048 --txqflags=0xf00 --disable-hw-vlan 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC			

7.4 Two VM Throughput (PHY-VM-VM-PHY)

Figure 7-14 shows the VM-VM test setup with 2 x 10 GbE ports (maximum 20 Gbps aggregate throughput) with packets being forwarded from the first VM to the second VM (and in the opposite direction). In comparison with PHY-VM-PHY with 1 VM and with 2 x 10GbE ports, there is an additional switching operation occurs in between 2 VMs for PHY-VM-VM-PHY. Therefore, instead of 2 switching operations in PHY-VM-PHY (with 2 x 10GbE ports), PHY-VM-VM-PHY has 3 switching operations.

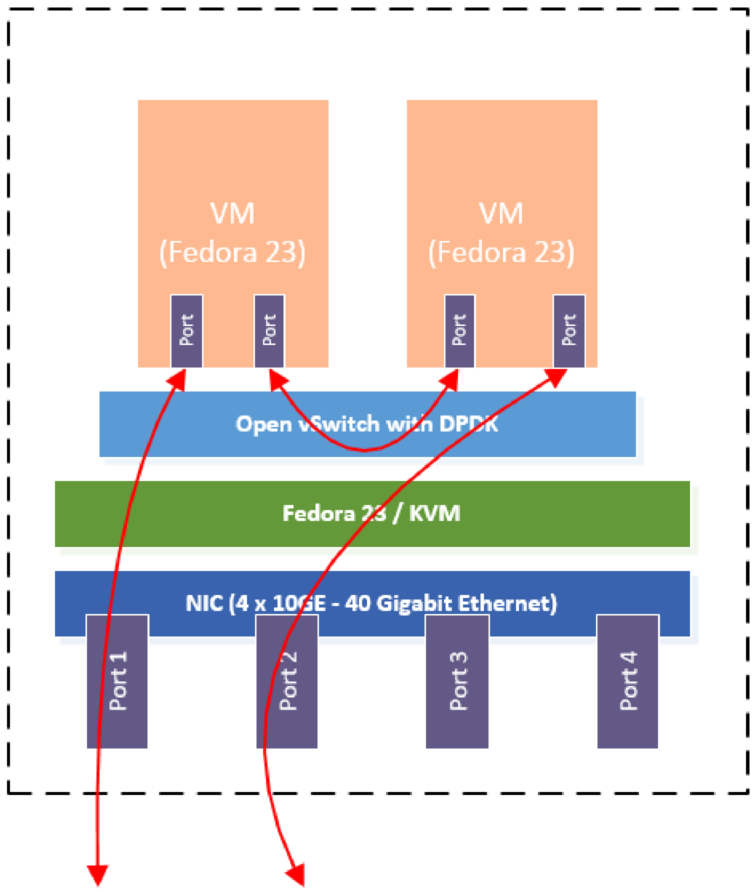

Figure 7-14 Two-VM Throughput performance test setup

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

This test compares two VM throughput with DPDK using 1 flow per port in each direction (total 2 flows).

Table 7-19 PHY-VM-VM-PHY test configuration variables for OvS with DPDK

Configuration variables	Ports	Flows per Port in each Direction	Physical Cores	Hyper-threaded cores
OvS with DPDK	2	1	1, 2,4	0, 2, 4

Figure 7-15 shows packet throughput with 2 flows (one flow per port in each direction) using 1 physical core, 1 hyper-threaded core, 2 physical cores, 2 hyper-threaded cores and 4 physical cores.

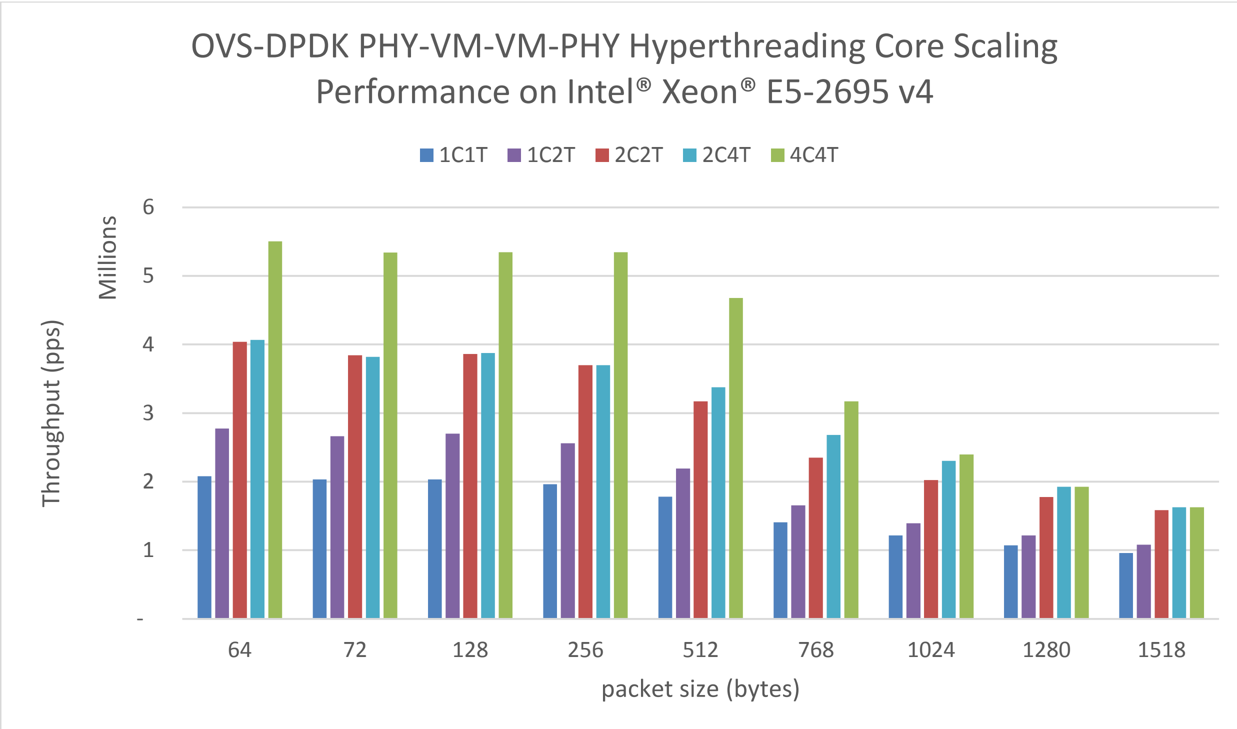

Figure 7-15 Two-VM Throughput (PHY-VM-VM-PHY) with 2 ports and 2 Flows

Table 7-20 shows throughput and latency data for each packet size for 4 hyper-threaded cores. Further scaling can be achieved by increasing number of cores.

Table 7-20 PHY-VM-VM-PHY test results for OvS with PDK (2 cores and 4 threads)

L3 Forwarding — Bidirectional Throughput with Zero Packet Loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (ns)	Minimum Latency (ns)	Maximum Latency (ns)
64	2,733	4,066,977	14	39,002	15,860	67,060
72	2,810	3,818,416	14	37,769	16,680	69,780
128	4,589	3,876,062	23	39,601	16,920	71,680
256	8,166	3,698,551	41	114,450	16,960	144,780
512	14,373	3,377,171	72	109,136	16,060	142,000
768	16,887	2,678,759	84	105,818	16,120	135,020
1024	19,246	2,304,343	96	102,703	16,960	150,200
1280	20,000	1,923,075	100	80,966	17,420	99,540
1518	20,000	1,625,487	100	60,980	17,420	78,300
Affinity Details	4PMD thread based OvS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=30000c On a VM: ./testpmd -c 0x6 -n 4 -- --burst=64 -i --txd=2048 --rxd=2048 --txqflags=0xf00 --disable-hw-vlan 2 Quad Port Intel® X710-DA4 NICs; 2 ports in use per each NIC					

7.5 40 Gbps Switching Performance

The goal of the tests is to determine the number of cores (number of PMD threads) to achieve 40 Gbps switching performance. Three different test scenarios were conducted to evaluate the number of OvS-DPDK PMD threads.

Test Scenario 1 is depicted in Figure 7-16. A total of 8 cores are needed to achieve 40 Gbps switching performance for test scenario 1 at 256 bytes packet size. Two cores are used for the switching, while 6 additional cores are used for passing the data to the VMs (i.e. for use by vHost and associated PMD). In this scenario each VM is transmitting and receiving 5 Gbps. The traffic over the wire is 20 Gbps transmit and 20 Gbps receive. Hence the OvS is switching 40 Gbps when accounting for the bidirectional nature of the traffic.

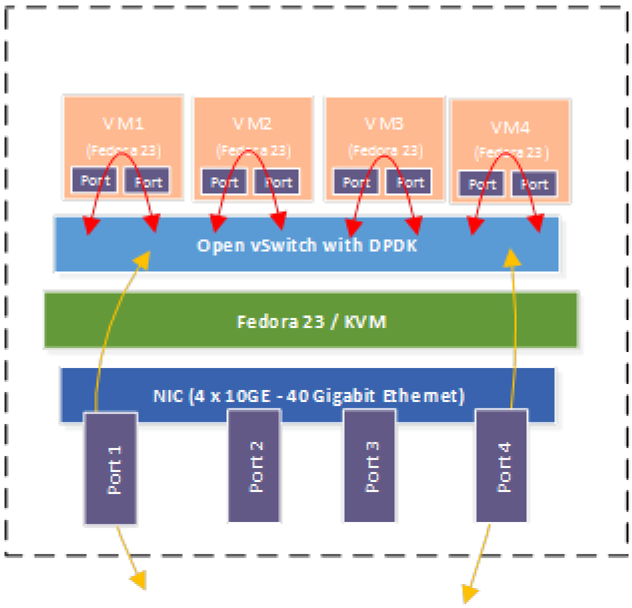

Figure 7-16 40 Gbps switching performance – test scenario 1

Test Scenario 2 is depicted in Figure 7-17. A total of 8 cores are needed to achieve 40 Gbps switching performance for test scenario 2 at 256 bytes packet size. This scenario is similar to Scenario 1, but some of the traffic is being switched between VM2 and VM3. The vSwitch is still doing 40 Gbps of switching as before, but with the additional VM to VM traffic, this will limit the throughput to 4 Gbps for each set of flows, with 16 Gbps on the wire.

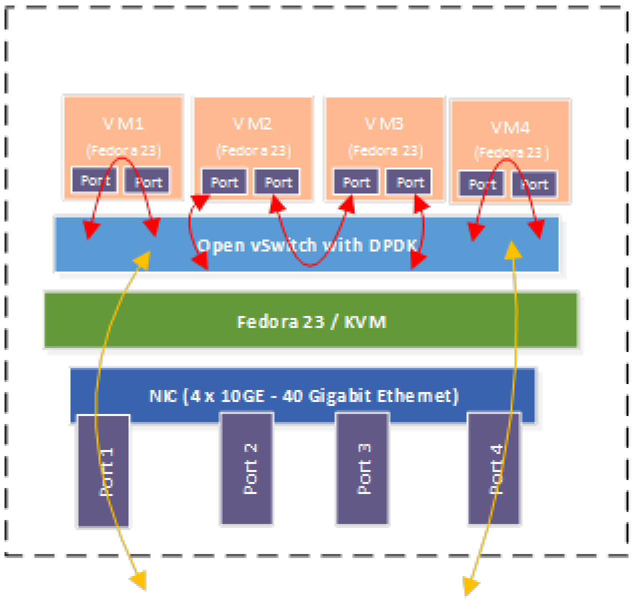

Figure 7-17 40 Gbps switching performance – test scenario 2

Test Scenario 3 is depicted in Figure 7-18. A total of 8 cores are needed to achieve 40 Gbps switching performance for test scenario 3 at 256 bytes packet size. This scenario is for a simple service function chain of two VMs.

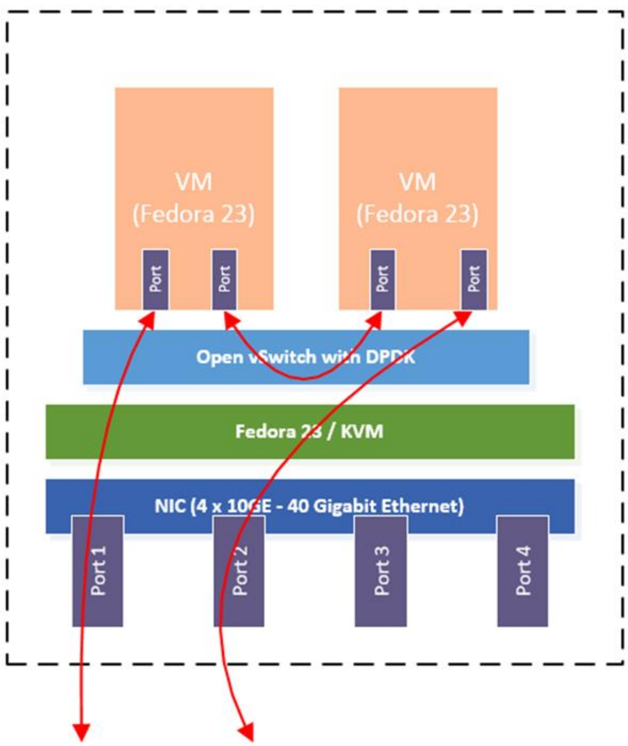

Figure 7-18 40 Gbps switching performance – test scenario 3

Figure 7-19 below shows results of core scaling for the 40 Gbps switching performance test with the use of three aforementioned test scenarios.

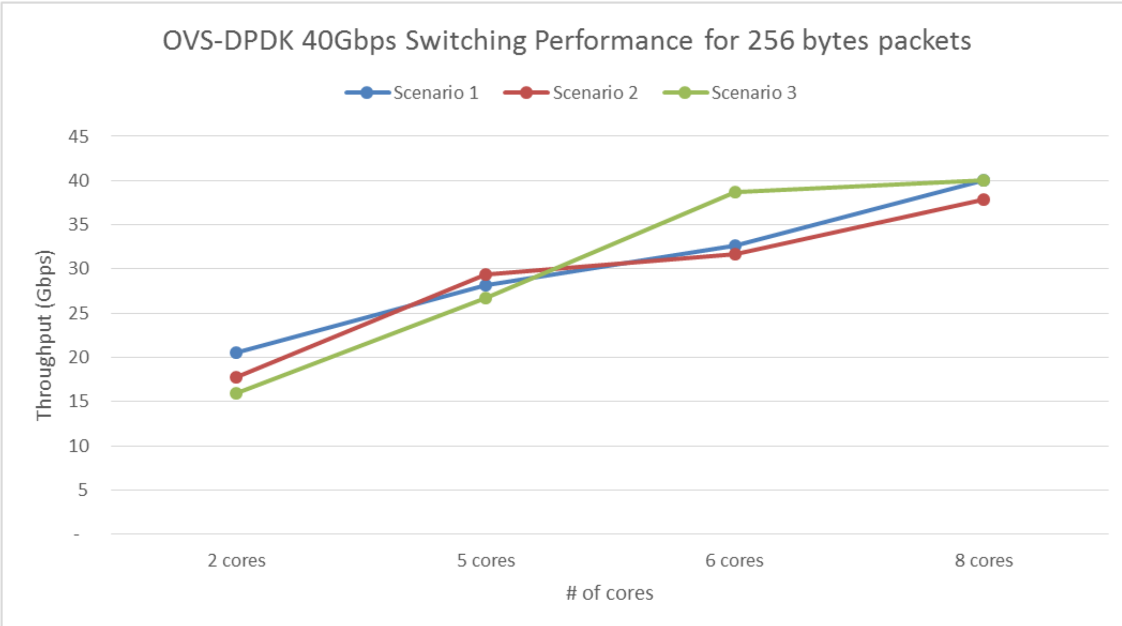

Figure 7-19 OvS with DPDK - 40 Gbps switching performance for 256B packets

Table 7-21 OvS with DPDK – 40 Gbps switching performance for 256B packets

L3 Forwarding — Bidirectional 256B 40Gbps switching performance with Zero Packet Loss						
Number of cores	Scenario 1		Scenario 2		Scenario 3	
	Mbps	Packets/sec	Mbps	Packets/sec	Mbps	Packets/sec
2	21,012	9,516,358	18,228	8,255,318	16,333	7,397,108
5	28,824	13,054,274	30,100	13,632,248	27,354	12,388,680
6	33,464	15,156,012	32,498	14,718,142	39,575	17,923,306
8	40,000	18,115,950	38,724	17,537,969	40,000	18,115,938
Affinity Details	2PMD, 5PMD, 6PMD and 8PMD thread based OvS and 0.0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=c (2 cores) ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=7c (5 cores) ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask= 807c (6 cores) ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=3807c (8 cores) On a VM: ./testpmd -c 0x6 -n 4 -- --burst=64 -i --txd=2048 --rxd=2048 --txqflags=0xf00 --disable-hw-vlan 2 ports from 1x Quad-Port Intel® X710-DA4 NIC					

7.6 VXLAN Performance (PHY-OVS-VM-OVS-PHY)

This test case investigates performance of VXLAN (https://tools.ietf.org/html/rfc7348) using native Open vSwitch* and Open vSwitch* with DPDK. The performance data provides a baseline for scenarios using VXLAN Tunnel End Points (VTEPs) in the vSwitch and establishes a test methodology for future comparisons. The test data here compare VXLAN performance for Native OvS and OvS with DPDK as well as VXLAN performance overhead for Native OvS and OvS with DPDK. The methodology described here attempts to emulate the scenario in Figure 7-20 that shows a VXLAN scenario using 1 x 10GbE port (maximum 10 Gbps aggregate throughput using two flows). VXLAN de-encapsulation and encapsulation processing occurs in the vSwitch VTEP.

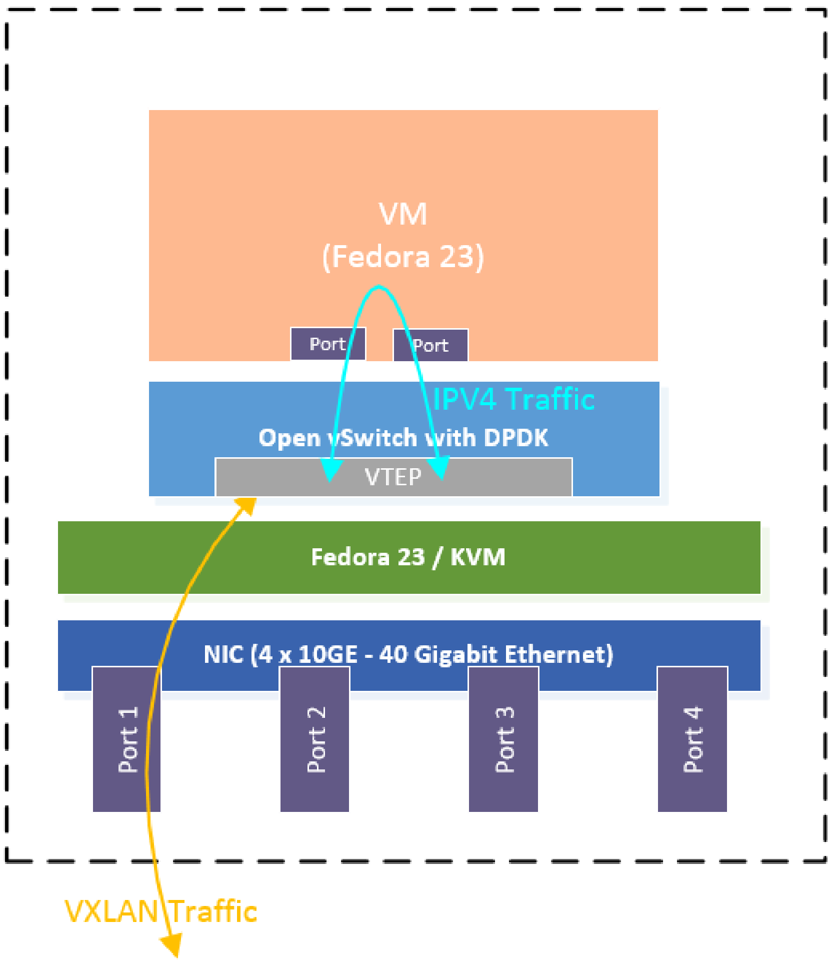

Figure 7-20 VXLAN Scenario with 1 Physical Port and VTEP in the vSwitch

7.6.1 VXLAN Test Methodology

In this test methodology, 1 port is connected to IXIA. IXIA generates packets with VXLAN header and send the VXLAN packets to the host (device under test). VTEP running inside the host de-encapsulates the packet received and send IPv4 packet to the VM and packets received from VM are encapsulated in VXLAN header by VTEP and sent them back to the IXIA.

The following steps show the flow of packets

1. IXIA generates VXLAN packets.
2. VXLAN packet is received by VTEP running inside the Host.
3. Host de-encapsulates the packet and sends it to VM.
4. VM forwards the packet to the VTEP.
5. VTEP running inside the Host encapsulates the packet by adding VXLAN header.
6. Packet with VXLAN header is sent to IXIA.
7. VXLAN encapsulation adds 50 byte header to each packet.

7.6.2 OvS and OvS with DPDK

VXLAN tests attempt to achieve system throughput of 10 Gbps using 1 physical ports and 1 unidirectional flow (see Table 7-22). Performance data shows comparison between:

- Native OvS and OvS with DPDK
- OvS with DPDK when using 1 and 2 physical cores.

Figure 7-21 shows VXLAN performance comparing native OvS and OvS with DPDK for 1 core and 2 cores configurations for all packet sizes. Throughput and latency data for 2 cores and hyper-threading off are provided in Table 7-23.

Table 7-22 VXLAN test configuration variables for native OvS and OvS with DPDK

Configuration variables	Ports	Flows per Port in each Direction	Physical Cores	Hyper-threaded cores
Native OvS	1	1	1	0
OvS with DPDK	1	1	1, 2	0

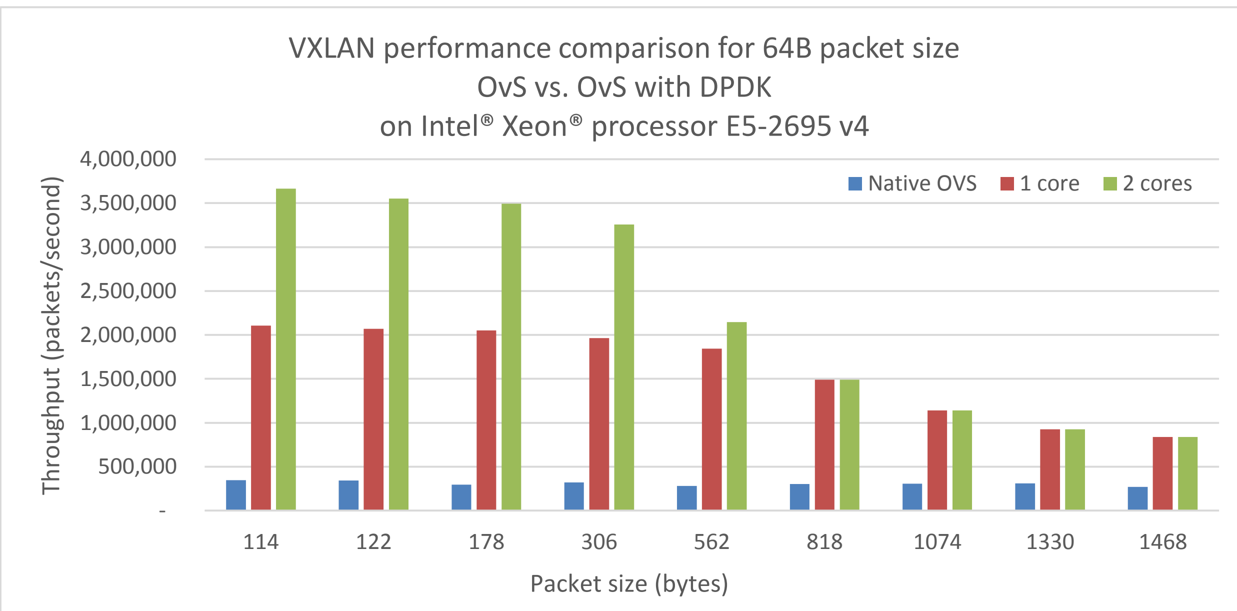

Figure 7-21 VXLAN Performance for 64-byte packets comparing native OvS and OvS with DPDK

Table 7-23 VXLAN Packet Throughput with 2 cores (hyper-threading off)

L3 Forwarding — Bidirectional Throughput with Zero Packet Loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (ns)	Minimum Latency (ns)	Maximum Latency (ns)
114	3,929	3,664,660	39	30,167	14,220	76,200
122	4,035	3,551,815	40	32,956	11,700	76,720
178	5,533	3,493,315	55	32,582	13,180	74,760
306	8,492	3,256,060	85	35,577	13,520	75,620
562	10,000	2,147,766	100	33,330	19,060	49,600
818	10,000	1,491,647	100	23,799	13,620	36,440
1074	10,000	1,142,596	100	21,110	13,080	33,380
1330	10,000	925,926	100	21,455	14,060	37,880
1468	10,000	840,054	100	19,743	12,920	32,500
Affinity Details	2PMD threads based OVS and 0% Loss resolution ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=C 1 port from 1x Quad-Port Intel® X710-DA4 NIC					

Figure 7-22 shows the impact of VXLAN overhead on the throughput compared to the PHY-VM-PHY test results using Native OvS and OvS with DPDK for small packet size (64 bytes). There is approximately 70% VXLAN performance overhead in Native OvS. Using OvS with DPDK the VXLAN performance overhead is significantly reduced: 39% VXLAN performance overhead with single core and ~47% VXLAN performance overhead with two cores are observed.

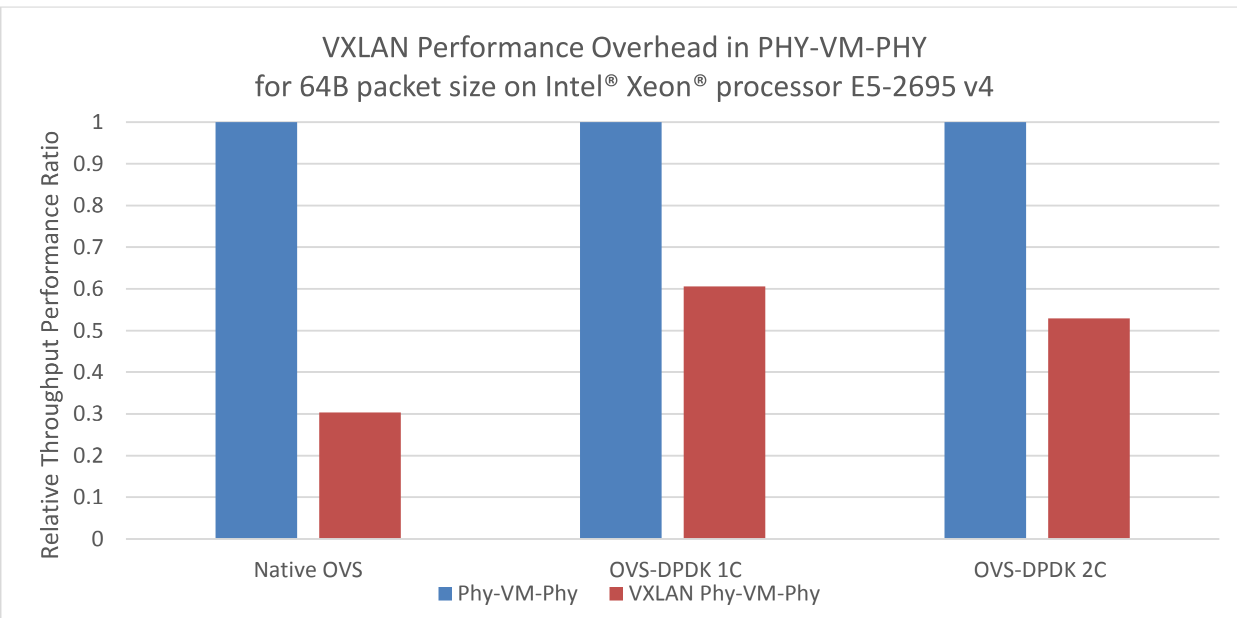

Figure 7-22 Impact of VXLAN overhead on processing performance in the PHY-VM-PHY test case for 64B packets

8.0 Industry Benchmarks

8.1 ETSI NFV

The European Telecommunications Standards Institute (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:

- Test methods for pre-deployment validation:
    - Validating physical DUTs and systems-under-test:
        - Data plane validation
        - Control plane validation
        - Management plane validation
    - Impact of virtualization on test methods
    - Considerations on choice of virtualized versus hardware based test appliances
- Pre-deployment validation of NFV infrastructure
- Pre-deployment validation of VNFs:
    - VNF life-cycle testing:
        - VNF instantiation testing
        - VNF termination
    - VNF data plane benchmarking
- Pre-deployment validation of network services
- Reliability & resiliency requirements
- Security considerations.

8.2 IETF

The Benchmark Working Group (BMWG) is one of the longest-running working groups in IETF. This group was re-chartered in 2014 to include benchmarking for virtualized network functions (VNFs) 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 (https://wiki.opnfv.org) is a carrier-grade, integrated, open-source platform to accelerate the introduction of new NFV products and services. As an open-source project, OPNFV aims 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.

Figure 8-1 illustrates the wide variety of performance test projects currently in OPNFV (this is a snapshot and evolving rapidly) and includes:

- infrastructure KPI verification
- platform performance benchmarking
- vSwitch performance
- system bottlenecks
- storage performance benchmarking
- controller performance.

For more information on test projects for OPNFV upcoming release (Brahmaputra) refer to OPNFV Wiki: https://wiki.opnfv.org/brahmaputra_testing_page.

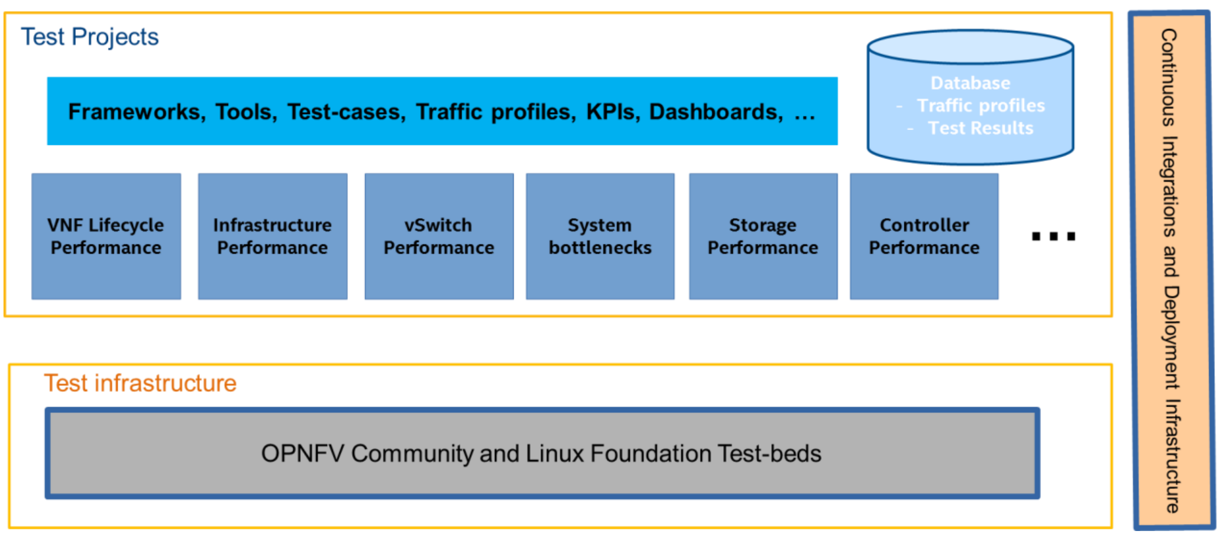

Figure 8-1 Snapshot of OPNFV Test projects and Infrastructure

The project "Characterize vSwitch performance for telco NFV Use Case" is highly relevant to this test report https://wiki.opnfv.org/characterize_vswitch_performance_for_telco_nfv_use_cases. A released Internet draft for benchmarking vSwitches in OPNFV is available https://tools.ietf.org/html/draft-vsperf-bmwg-vswitch-opnfv-00. The Internet draft describes the progress of the OPNFV project on vSwitch performance with additional considerations when vSwitches are implemented in general-purpose hardware. The project intends to build on the current and completed work of the Benchmarking Working Group in IETF.

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 the 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

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 created only 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_ts
o6=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_ts
o6=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 process ID (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 in runtime for each QEMU child thread and determine the active QEMU threads:

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

Output:

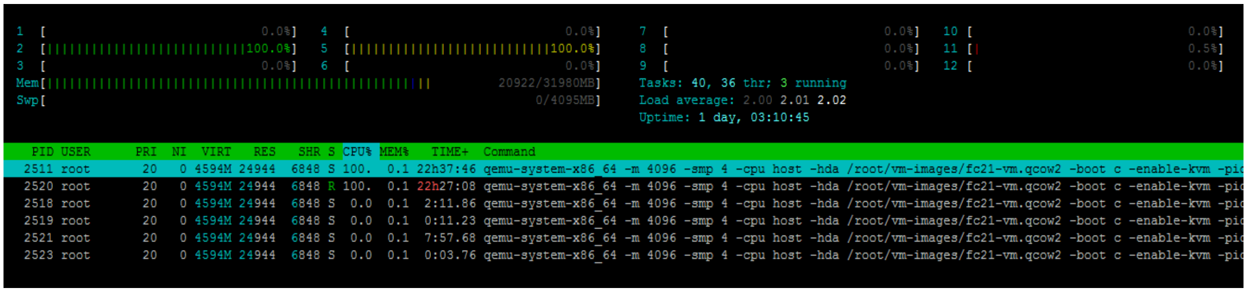

Figure 9-1 Output from htop showing high CPU usage for active QEMU threads

From the `htop` output screen, you can view two active QEMU threads that have a 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 do not use core-affinity, 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 media access control (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 that are 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 Address Resolution Protocol (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 OvS Test Setup

10.1 Configure the Host Machine

1. Stop and disable the interruption request (IRQ) balance:

```
# killall irqbalance
# systemctl stop irqbalance.service
# systemctl disable irqbalance.service
```

2. Stop and disable the Firewall and iptables:

```
# systemctl stop firewalld.service
# systemctl disable firewalld.service
# systemctl stop iptables.service
```

3. Disable Security-enhanced Linux (SELinux):

```
[root@localhost ~]# vi /etc/selinux/config
SELINUX=disabled
```

4. Disable 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
```

5. Disable 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
```

6. 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*:

```
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=1-17,19-35 rhgb quiet"
```

2. With hyper-threading disabled, 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=16 hugepagesz=2M hugepages=2048
intel_iommu=off isolcpus=1-17 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-*
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 2.2

1. Go to the DPDK-2.2.0 directory and run the following:

```
# make install T=x86_64-native-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
```

10.4 Install OvS

1. Go to the OvS directory and run:

```
# ./boot.sh
# ./configure --with-dpdk=/root/dpdk-2.2.0/x86_64-native-linuxapp-gcc \
               CFLAGS=”-Ofast -g”
# make 'CFLAGS=-g -Ofast'
```

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:

```
# mount
nodev on /mnt/huge type hugetlbfs (rw,relatime)
```

```
nodev on /mnt/huge_2mb type hugetlbfs (rw,relatime,pagesize=2MB)
```

3. Remove the following Linux modules and 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:

```
# 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)
01:00.2 Ethernet controller: Intel Corporation Ethernet Controller X710 for
10GbE SFP+ (rev 01)
01:00.3 Ethernet controller: Intel Corporation Ethernet Controller X710 for
10GbE SFP+ (rev 01)
```

10.6 Bind 10 GbE NIC Ports to the igb_uio Driver

1. 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 01:00.2
# python $DPDK_DIR/tools/dpdk_nic_bind.py --bind=igb_uio 01:00.3
# python $DPDK_DIR/tools/dpdk_nic_bind.py -status
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
0000:01:00.2 'Ethernet Controller X710 for 10GbE SFP+' drv=igb_uio unused=i40e
0000:01:00.3 '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
```

10.1 Initialize the New OvS Database

1. 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
```

2. Start the database server:

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

3. Initialize the OvS database:

```
# ./utilities/ovs-vsctl --no-wait init
```

10.2 Start OvS-vSwitchd

1. 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
```

10.3 Tune OvS-vswitchd

You can check the thread siblings list (when hyper-threading 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

1. 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

1. 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
2,9
```

2. Then set the pmd-cpu-mask to CPU2 and CPU10 (0x404):

```
# ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=404
# ./ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

3. 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

1. 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
2,9
# cat /sys/devices/system/cpu/cpu3/topology/thread_siblings_list
3,10
```

2. Then set the pmd-cpu-mask to CPU2, CPU3, CPU10, and CPU11 (0xC0C).

```
# ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask= C0C
# ./ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

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

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

10.4 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
```

10.5 Add the Port Flows

1. Clear current flows:

```
# export OVS_DIR=/root/ovs
# cd $OVS_DIR
# ./utilities/ovs-ofctl del-flows br0
```

2. 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 up to the section 10.3 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

1. Clear current flows

```
# export OVS_DIR=/root/ovs
# cd $OVS_DIR
# ./utilities/ovs-ofctl del-flows br0
```

2. 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

1. 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 3 -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_ts
o6=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_ts
o6=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_ts
o6=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_ts
o6=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 the 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-*
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

1. 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 2.2

1. Download DPDK 2.2.0 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-2.2.0/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-2.2.0
# 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-2.2.0/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-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:04.0
# /root/dpdk-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:05.0
# /root/dpdk-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:06.0
# /root/dpdk-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:07.0
# /root/dpdk-2.2.0/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-2.2.0/x86_64-native-linuxapp-gcc/build/app/test-pmd
# ./testpmd -c 0x3 -n 4 -- --burst=64 -i --disable-hw-vlan --txd=2048 \
          --rxd=2048 --txqflags=0xf00
```

2. In the application, enter the `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=64
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 9.5 CPU Core Affinity for the Virtual Machine (qemu-kvm), to set the active QEMU threads to the correct core affinity.

Table 11-1 CPU Affinity Setting on the Host

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

Table 11-2 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 10.0 OvS Test Setup and follow up to the section 10.3 Tune OvS-vswitchd, to set up the host configurations, 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

1. Clear current flows

```
# export OVS_DIR=/root/ovs
# cd $OVS_DIR
 # ./utilities/ovs-ofctl del-flows br0
```

2. 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

1. Start the first VM on CPU 3, CPU 4, and CPU 5 (0x38) with the following configuration:

```
# taskset 38 qemu-system-x86_64 -m 4096 -smp 3 -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_ts
o6=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_ts
o6=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 the 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-*
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 5, CPU6, and CPU7 (0xE0) with the following command:

```
# taskset E0 qemu-system-x86_64 -m 4096 -smp 3 -cpu host -hda /root/vm-
images/vm2-fc21.img -boot c -enable-kvm -pidfile /tmp/vm2.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_ts
o6=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_ts
o6=off,guest_ecn=off,mrg_rxbuf=off \
```

```
--nographic -vnc :15
```

12.4 Set up the VM HugePages

1. 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 2.2

1. Download DPDK 2.2.0 and compile it:

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

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

```
# vi /root/dpdk-2.2.0/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-2.2.0
# 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-2.2.0/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 the igb_uio driver:

```
# /root/dpdk-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:04.0
# /root/dpdk-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:05.0
# /root/dpdk-2.2.0/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 the test-pmd app on vCPU1 and vCPU2 (0x6):

```
# cd /root/dpdk-2.2.0/x86_64-native-linuxapp-gcc/build/app/test-pmd
# ./testpmd -c 0x3 -n 4 -- --burst=64 -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=64
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 9.5 CPU Core Affinity for the Virtual Machine (qemu-kvm), to set the active QEMU threads to the correct core affinity.

Table 12-1 CPU affinity setting on the host

Logical Core	Process
1	ovs-vswitchd
2	PMD0
3, 4, 5	QEMU (VM1)
5, 6, 7	QEMU (VM2)

Table 12-2 VM1 QEMU threads CPU affinity

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

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

Table 12-3 VM2 QEMU threads CPU affinity

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

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

13.0 VXLAN Test Setup

Follow the instructions below to configure VXLAN test setup. Test setup configurations include using native OvS and OvS with DPDK.

13.1 Native OvS Setup

To setup and start regular OvS in Host please refer to section 10.1 Configure the Host Machine and follow the instructions below.

13.1.1 Set the Kernel Boot Parameters

1. With hyper-threading enabled 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=1-7 rhgb quiet"
```

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

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

3. 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-*
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
```

13.1.2 Compile and Install OvS

1. Go to the OvS directory and run:

```
# ./boot.sh
# ./configure
# make
# make install
# ./configure --with-linux=/lib/modules/4.2.3-300.fc23.x86_64/build \
               CFLAGS="-Ofast -g"
# make 'CFLAGS=-g -Ofast'
```

13.1.3 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:

```
# mount
nodev on /mnt/huge type hugetlbfs (rw,relatime)
nodev on /mnt/huge_2mb type hugetlbfs (rw,relatime,pagesize=2MB)
```

3. Load the modules:

```
# modprobe openvswitch
# modprobe i40e
```

4. 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
```

5. Initialize the new OvS database and start the server:

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

6. Start the database server:

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

7. Initialize the OvS database:

```
# ./utilities/ovs-vsctl --no-wait init
```

8. Start OvS-vswitchd:

```
# ./vswitchd/ovs-vswitchd
```

13.1.4 Create the Ports and VXLAN VTEP

1. Create the VXLAN tunnel between 2 hosts:

```
# ./utilities/ovs-vsctl add-br br0
# ifconfig br0 2.2.2.1/24
# ./utilities/ovs-vsctl add-port br0 eth3
# ./utilities/ovs-appctl ovs/route/add 2.2.2.2/24 br0
```

2. Create an internal bridge:

```
# ./utilities/ovs-vsctl add-br br-int
```

```
# ifconfig br-int 1.1.1.1/24
# tunctl -d tap1
# tunctl -d tap2
# tunctl -t tap1 -u root
# tunctl -t tap2 -u root
 ifconfig tap1 0
 ifconfig tap2 0
#Add the Tap devices to the OVS Bridge
# ./utilities/ovs-vsctl add-port br-int tap1
# ./utilities/ovs-vsctl add-port br-int tap2
```

3. Add VXLAN VTEP:

```
# ./utilities/ovs-vsctl add-port br-int vxlan0 -- set Interface \
          vxlan0 type=vxlan options:remote_ip=2.2.2.2 options:key=1000
# ./utilities/ovs-vsctl show
# ./utilities/ovs-appctl ovs/route/show
```

13.1.5 Add the Port Flows

1. Clear current flows:

```
# cd $OVS_DIR
# ./utilities/ovs-ofctl del-flows  br-int
```

2. Add flow for port 1 (physical) to port 2 (VTEP):

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

13.1.6 Power on the VM

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

```
# taskset 70 qemu-system-x86_64 -cpu host -boot c -m 4096M -smp 3 -hda
/root/vm-images-fedora/Fed23_VM.img --enable-kvm -name 'VM1' -vnc :1 -pidfile
/tmp/vm1.pid --nographic -monitor unix:/tmp/vm1monitor,server,nowait  -object
memory-backend-file,id=mem,size=4096M,mem-path=/mnt/huge,share=on -numa
node,memdev=mem -mem-prealloc -net none -netdev
type=tap,id=net1,script=no,downscript=no,ifname=tap1,vhost=on -device virtio-
net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_ts
o6=off,guest_ecn=off,mrg_rxbuf=off -netdev
type=tap,id=net2,script=no,downscript=no,ifname=tap2,vhost=on -device virtio-
net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_ts
o6=off,guest_ecn=off,mrg_rxbuf=off &
```

13.1.7 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 the 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-*
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
```

13.1.8 Set up the VM HugePages

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

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

13.1.9 Set up the vHost Network in the VM

1. Load the UIO kernel module in the VM:

```
# modprobe uio
# insmod /root/dpdk-2.2.0/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 the igb_uio driver:

```
# /root/dpdk-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:04.0
# /root/dpdk-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:05.0
# /root/dpdk-2.2.0/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>
```

13.1.10 Start test-pmd Application in the VM

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

```
# cd /root/dpdk-2.2.0/x86_64-native-linuxapp-gcc/build/app/test-pmd
# ./testpmd -c 0x3 -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
```

13.2 OvS with DPDK Setup

To set up and start OvS with DPDK in Host refer to section 10.0 OvS Test Setup and follow up to the section 10.2 Start OvS-vSwitchd. Then follow the instructions below to configure the VXLAN test setup.

13.2.1 Tune OvS-vSwitchd for VXLAN

Once the OvS-vSwitchd is running, we setup the CPU core affinity for the OvS PMD threads to 1 core, and 2 cores respectively.

One-PMD Configuration

1. 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
```

Two-PMD Configuration

1. 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
```

13.2.2 Create the Ports and VXLAN VTEP

1. Create the VXLAN tunnel between 2 hosts:

```
# ./utilities/ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
# ifconfig br0 2.2.2.1/24
# ./utilities/ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk
# ./utilities/ovs-appctl ovs/route/add 2.2.2.2/24 br0
```

2. Create an internal bridge:

```
# ./utilities/ovs-vsctl add-br br-int
# ifconfig br-int 1.1.1.1/24
# ./utilities/ovs-vsctl add-port br-int vhost-user0 -- set Interface vhost-
user1 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br-int vhost-user1 -- set Interface vhost-
user2 type=dpdkvhostuser
```

3. Add VXLAN VTEP:

```
# ./utilities/ovs-vsctl add-port br-int vxlan0 -- set Interface \
           vxlan0 type=vxlan options:remote_ip=2.2.2.2 options:key=1000
# ./utilities/ovs-vsctl show
# ./utilities/ovs-appctl ovs/route/show
```

13.2.3 Add the Port Flows

1. Clear current flows:

```
# cd $OVS_DIR
# ./utilities/ovs-ofctl del-flows  br-int
```

2. Add flow for port 1 (physical) to port 2 (VTEP):

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

13.2.4 Power on the VM

1. 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 3 -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_ts
o6=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
```

13.2.5 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 the 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-*
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
```

13.2.6 Set up the VM HugePages

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

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

13.2.7 Set up the vHost Network in the VM

1. Load the UIO kernel module in the VM:

```
# modprobe uio
# insmod /root/dpdk-2.2.0/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 the igb_uio driver:

```
# /root/dpdk-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:04.0
# /root/dpdk-2.2.0/tools/dpdk_nic_bind.py -b igb_uio 00:05.0
# /root/dpdk-2.2.0/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>
```

13.2.8 Start test-pmd Application in the VM

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

```
# cd /root/dpdk-2.2.0/x86_64-native-linuxapp-gcc/build/app/test-pmd
# ./testpmd -c 0x3 -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
```

Appendix A: Acronyms and Abbreviations

Abbreviation	Description
ARP	Address Resolution Protocol
BMWG	Benchmark Working Group
CPU	Central Processing Unit
DPDK	Data Plane Development Kit
DUT	Device-Under-Test
EMC	Exact Match Cache
ETSI	European Telecommunications Standards Institute
GbE	Gigabit Ethernet
GRUB	GRand Unified Bootloader
IETF	Internet Engineering Task Force
IPDV	Inter-Packet Delay Variation
IPv4	Internet Protocol version 4
IRQ	Interruption Request
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
MAC	Media Access Control
NFV	Network Functions Virtualization
NIC	Network Interface Card
NUMA	Non-Uniform Memory Access
ONP	Intel® Open Network Platform

Abbreviation	Description
OPNFV	Open Platform for NFV
OvS	Open vSwitch
PCI	Peripheral Component Interconnect
PDV	Packet Delay Variation
PHY	Physical Layer
PID	Process ID
PMD	Poll Mode Driver
QEMU	Quick Emulator
RFC	Request for Comments
SDN	Software-Defined Networking
SELinux	Security-Enhanced Linux
SLA	Service-Level Agreement
TLB	Translation Lookaside Buffer
vCPE	Virtual Customer Premises Equipment
vhost	Virtual Host
VM	Virtual Machine
VNF	Virtualized Network Function
VTEP	VXLAN Tunnel End Point
VXLAN	Virtual eXtensible LAN

Appendix B: References

Title	Reference
01.org: Intel® Open Network Platform	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: Intel® ONP 2.1 Reference Architecture Guide	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: Intel® ONP 2.1 Release Notes	https://01.org/packet-processing/intel%C2%AE-onp-servers
Intel® Ethernet Converged Network Adapter X710-DA4	https://ark.intel.com/products/83965/Intel-Ethernet-Converged-Network-Adapter-X710-DA4?q=X710%20DA4
Intel® Xeon® processor E5-2695 v4	https://ark.intel.com
RFC 2544 Benchmarking Methodology for Network Interconnect Devices	https://tools.ietf.org/html/rfc2544
RFC 2679 A One-way Delay Metric for IPPM	https://tools.ietf.org/html/rfc2679
RFC 3393 IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)	https://tools.ietf.org/html/rfc3393
RFC 3432 Network performance measurement with periodic streams	https://tools.ietf.org/html/rfc3432
RFC 3550 RTP: A Transport Protocol for Real-Time Applications	https://tools.ietf.org/html/rfc3550
RFC 5481 Packet Delay Variation Applicability Statement	https://tools.ietf.org/html/rfc5481
RFC 7348 Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks	https://tools.ietf.org/html/rfc7348
Supermicro X10DRH-I	https://www.supermicro.com/products/motherboard/Xeon/C600/X10DRH-i.cfm

Legal Information