Intel® Open Network Platform Release 2.0 vE-CPE Performance Test Report

SDN/NFV Solutions with Intel® Open Network Platform

Document Revision 1.0
January 2016

Revision History

Date	Revision	Comments
January 22, 2016	1.0	Initial document for release of Intel® Open Network Platform Release 2.0

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 available on 01.org (e.g. Intel® Open Network Platform Reference Architecture Guides, Performance Test Reports).

The primary audiences for this test report are architects and engineers implementing virtual enterprise customer premises equipment (vE-CPE) on Intel® Architecture and open-source software elements including:

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

This test report provides a guide to packet processing performance for vE-CPE use case with the Intel® ONP i.e. for network services hosted on a virtualized server located at the customer premises. The report includes information on relevant performance metrics and test methodologies, as well as discussion on vE-vCPE performance requirements.

The configurations and methods used do not imply a single “correct” approach, however detailed setup guides and test results provide a reference for architects and engineers who are evaluating and implementing SDN/NFV solutions with a goal to achieve optimal system performance. The summary and the analysis provided can also help technical stakeholders evaluate vE-CPE solutions on the low cost Intel® Atom™ Processor C2000 Family-based platforms.

2.0 Summary

Evaluating realistic performance and functionality of NFV use-case deployment scenarios includes integrating configuration and provisioning systems as well as using established tools and test methods. Intel® ONP uses OpenStack as the Virtual Infrastructure Manager and this was used to provision the vE-CPE use case. Test results from scenarios that use management tools such as OpenStack are not comparable to those that are manually setup and configured. This report provides detailed configuration information for Linux, Open vSwitch*, DPDK, and OpenStack*.

NFV technologies and methodologies are rapidly evolving and it is therefore critical to understand relevant Open-Source project roadmaps and current gaps. To leverage Open-Source communities a prime goal must be to automate all aspects of the NFV solution lifecycle (i.e. build, deploy and test).

2.1 Delivering Services with vE-CPE

Service Provider networks today are generally comprised of proprietary hardware and software. Management and flexibility are constrained by fixed-function boxes, limiting the ability to quickly deploy services and meet new demands. Services delivered using customer premises equipment (CPE) have the added complexity of deploying and managing equipment located on the customer premises. Network elements built using Open Source software running on standard high volume servers offers the promise of lower total cost and much greater flexibility for managing services and rapidly trialing new services.

vE-CPE targets Small to Medium Enterprises who typically outsource Network Management functions. vE-CPE is characterized by an Enterprise edge device (i.e. at the customer premises) and based on standard platform architecture, provisioned/configured remotely.

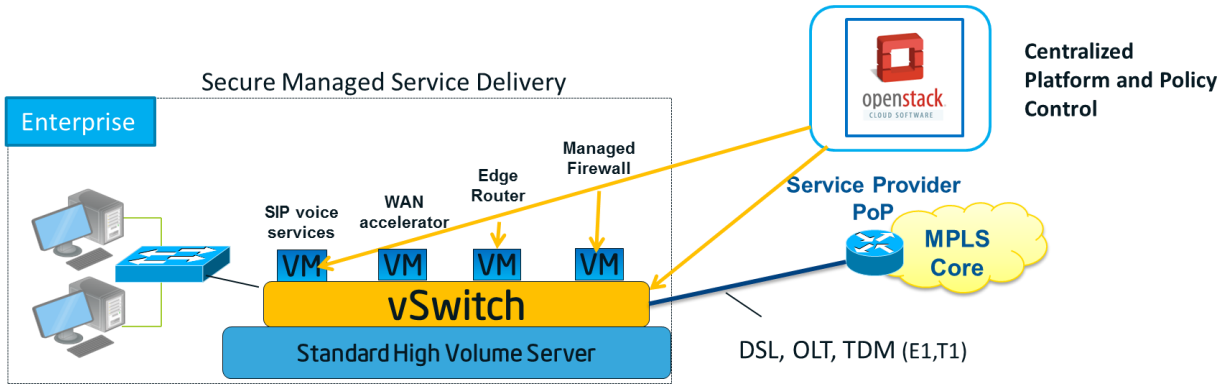

Figure 2-1 Depiction of vE-CPE use-case showing virtualized edge device (on left) that is managed with Open Stack and Open Daylight hosted in the Service Provider network.

Examples of enterprise CPE network services include:

- Managed wide area network (WAN) services where the CPE acts as an enhanced customer edge router to deliver e.g. multiprotocol label switching (MPLS) connectivity and other Ethernet WAN services.
- SIP based (VoIP) voice services.
- VPN services for remote workers.
- Security services such Firewall, DPI and other intrusion prevention services (IPS).

2.2 Use of OpenStack

As mentioned above OpenStack was used as the Virtual Infrastructure Manager to configure the vE-CPE use case. OpenStack – Neutron was used to provision the OvS fabric connecting VMs with each other and the external networks.

2.3 Hardware Platform

Tests were conducted using a single socket SuperMicro System-on-Chip (SoC) platform with Intel® Atom™ Processor C2758 (formerly Rangeley) http://ark.intel.com/products/77988/Intel-Atom-Processor-C2758-4M-Cache-2_40-GHz. This processor has 8 cores, base processor frequency of 2.4 GHz and TDP of 20W. Two ports of the integrated 1 GbE LAN were used for all tests.

2.4 Test Setup

For characterizing data-plane performance the vE-CPE (system-under-test) was configured with 3 VNFs using OpenStack to deploy VMs and configure the internal vE-CPE network. Note that the vE-CPE use case typically requires VMs be provisioned remotely but not to download the images on reset or startup. The VLAN protocol was used as the overlay network with OpenStack providing the networks on the vE-CPE.

For optimized data path performance it is common to use Data-Plane Development Kit (DPDK) based VNFs. Note that OpenStack-Neutron deploys the Open vSwitch network bridge setup with normal action flows mandating the vSwitch to learn the network port information for the VNFs being deployed. Provided that current DPDK based network ports do not respond to ARP packets, the benchmarking methodology then requires the vSwitch to first learn about the VNF's network ports before instantiating the DPDK based application. Explanation of the test methodology explains details of using OvS with DPDK based VNFs.

The test harness is implemented using IxNetwork* traffic generator which emulates an appropriate network topology for communicating with VMs in the system-under-test.

2.5 Measuring Throughput

Tests in this report are most relevant to vE-CPE supporting up to a few hundred users with aggregate public Internet bandwidth less than 100 Mbps (WAN) and an customer internal LAN of 1 Gbps (LAN).

In the seven-layer OSI model of computer networking, a packet refers to a data unit at layer 3 (network layer), a frame refers to a data unit at layer 2 (data link layer), and a segment or datagram refers to a data unit at layer 4 (transport layer).

Throughput is defined in RFC 1242 as the maximum rate at which none of the offered frames are dropped by the device. Throughput is often expressed as "frame rate" (measured in frames per second (fps) or packets per second (pps). An alternative measure is percent of line-rate.

Frame rate is based on the bit rate and frame format definitions. All test configurations in this report use 1 GbE ports and line-rate refers to a MAC bit rate of 1 billion bits per second as defined by the IEEE 802.3 standard. The frame rate for 1 GbE is determined by dividing the line-rate by the preamble + frame length + interframe gap.

2.6 Measuring Jitter

RFC 3393 provides definitions of Packet Delay Variation (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.

In this document the term “jitter” is used generically however for purpose of measurements it should be taken to mean PDV.

3.0 vE-CPE Requirements

As a starting point, the reader is referred to the relevant ETSI NFV use-case – Virtual Network Function as a Service (VNFaaS) http://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf.

3.1 vE-CPE Workloads

Some typical workloads include:

- switching and routing for packet forwarding
- session control (SBC) for VoIP security
- quality of service and admission control
- firewall and DPI for security
- WAN accelerator for traffic optimization
- encryption/decryption for VPN

A bottom-up approach to vE-CPE performance characterization is to decompose workloads into resource intensive (i.e. compute, I/O, storage) operations that can be tested individually in order to understand the bottlenecks and performance limits. This is illustrated in the table below.

Table 3-1 vE-CPE can be decomposed into a number workload operations for evaluating performance

		Packet Operation							
		L2/L3 Port forwarding	Pattern match	ACL	Encrypt / decrypt	H-QoS	Encap/ decap	Compression	Disc caching
vE-CPE workload	Switching / Routing	✓	✓				✓		
	SBC		✓			✓			
	Firewall / DPI		✓	✓					
	WAN accelerator							✓	✓
	VPN				✓				

Note: The initial version of this report only considers port forwarding as the VNF packet operation.

3.2 Traffic Profile

For testing we assume a homogenous and steady-state traffic profile with data traffic and voice traffic packet characteristics. Data traffic metrics are primarily concerned with throughput while key traffic metric for voice are "latency" and "jitter". We consider these two profiles independently.

3.2.1 Packet Size

Internet Mix (IMIX) is often used to represent data traffic; see https://en.wikipedia.org/wiki/Internet_Mix. 64B packets are used for simulating voice traffic as a worst case condition. A narrow-band codec widely used for voice traffic such as G.711 transmits at 64Kbps with a corresponding default payload size 160 Bytes.

Initial tests in this report used fixed length packet sizes from 64Bytes to 1518 Bytes.

3.2.2 Flows

If the maximum number of users (endpoints) is 250, each running 3 or 4 applications, the maximum of bi-directional flows may be between 750 and 1000. We assume the following flow characteristics:

- Up to 1000 simultaneous UDP bidirectional flows (i.e. 1000 flows in each direction).
- Flow tables are fully populated and in steady-state (there is no delay communicating with the network controller and routes have converged).

3.3 Throughput

Typical access technologies used for CPE services include EFM (Ethernet in the first mile), xDSL (Digital Subscriber Line), VSAT (small terminal satellites), and legacy TDM (E1, T1). For CPE services today 100 Mbps WAN is considered top end. Even with VDSL typical speeds are 40 Mbps down-link and 10 Mbps up-link.

Assuming a 100 Mbps WAN link, the maximum theoretical throughput rates are:

- Each VM processes 200 Mbps (100 Mbps in each direction).
- With 3 VMs the vSwitch might be expected to process 800Mbps aggregate packet throughput (400 Mbps in each direction).

Table 3-2 Maximum Throughput for 1 GbE

IP Packet Size (Bytes)	Bidirectional line-rate (fps)	Unidirectional line-rate (fps)
64	2,976,192	1,488,096
128	1,689,190	844,595
256	905,798	452,899
512	469,926	234,963
1024	239,464	119,732
1280	192,308	96,154
1518	162,550	81,275

3.4 Packet Loss

For voice traffic acceptable packet loss depends on the codec used as some adapt better than others and it also depends over what period packet loss is measured.

The G.729 codec often used in VoIP applications where bandwidth must be conserved (such as conference calls) and requires packet loss significantly less than 1 percent to avoid audible errors. Total packet loss of a few percent could be tolerable if evenly distributed over a call but would be unacceptable if measured over 1 or 2 seconds.

Generally, every packet should be accounted for when characterizing network equipment.

3.5 Latency (Packet Delay)

End-to-end latency budget rather than any element in isolation impacts customer experience and depends on codec and context (e.g. person to person or calling into conference bridges).

Worst case latency must be considered. Outliers can impact customer experience and ultimately cost the service provider. Latency tests should be run for at least 1 hour or preferably 24h. For example a very brief 1 second latency with older VoIP codecs can set the jitter buffer to about 1.5 seconds creating a permanent unacceptable delay on the voice call of ~1.5 seconds.

Circuit voice latency in commercial networks is typically 200 – 300 ms which provides for smooth conversation using public switched telephone network (PSTN). For latency in general, mouth-to-ear latency of less than 150 ms is considered acceptable (ITU-T G.114 recommends a maximum of a 150 ms one-way latency). User experience will begin to deteriorate significantly above 250 ms. This includes the entire voice path, part of which may be on the public Internet and it includes encoding time so that needs to be added into the end-to-end budget. Therefore a network element should have latency budget of considerably less than 150 ms. Measuring VNF delay in the sub millisecond is ideal.

3.6 Measuring Throughput and Latency

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 such as 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 smallest frames of 64 bytes to largest frames of 1518 bytes.

RFC 2544 types of tests are:

- Throughput test defines the maximum number of frames per second that can be transmitted without any error. Test time during which frames are transmitted must be at least 60 seconds.
- Latency test measures the time required for a frame to travel from the originating device through the network to the destination device.
- Frame loss test measures 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 will rapidly degrade service quality.
- Burst test assesses the buffering capability of a switch. It measures the maximum number of frames received at full line rate before a frame is lost. In carrier Ethernet networks, this measurement validates the excess information rate as defined in many SLAs.
- System recovery to characterize speed of recovery from an overload condition
- Reset to characterize speed of recovery from device or software reset

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.

3.6.1 IXIA Considerations for Measuring Throughput

Ixia by default adds additional 20 bytes to the configured packet size while transmitting across the physical interface. However the calculated throughput displayed by Ixia statistics, IxNetwork* for the purposes of this report, does not consider the additional 20 byte addition by the traffic generator. Hence the throughput in Megabits per second (Mbps) indicated in this document is calculated using the below mentioned formula:

Throughput in Mbps = (Configured Packet Size +20)*8*Displayed Frames per Second (fps) / 1,000,000. Correspondingly, % line rate is calculated in this document using the formula:

% line rate = (Mbps*100)/(Number of flows * Theoretical Max of each port).

Note that the minimum packet size indicated in the test results is 66 Bytes for plain traffic and 70 Bytes for VLAN traffic, with all the RFC 2544 benchmark measurements. This is due to enabling the 4 Byte signature option in the IxNetwork* tool while configuring RFC 2544 methodology.

3.7 Jitter (Packet Delay Variation)

For VoIP adequate jitter buffers are required to manage quality which further adds to the end-to-end latency, and are usually only effective on delay variations less than 100 ms. Jitter must therefore be minimized and should be measured in sub millisecond range to be acceptable. Maximum latency defines the maximum possible jitter, which can vary between zero and max latency.

3.8 Measuring Packet Delay Variation (PDV)

The RFC 2544 measurement of latency is extensively used in traditional testing. With NFV we need more information on latency including packet delay variation. In principle we would like to know the delay of all packets, but in practice some form of sampling is needed (may not be periodic sampling).

The results needed are the distribution of delays under various conditions, which will in turn help to determine next sampling strategies and appropriate statistical summarization of the variation.

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" however this term causes confusion because it is used in different ways by different groups of people. For this reason we avoid using the term “jitter”. The ITU-T also recommends various delay variation metrics [Y.1540] [G.1020]. Most of these standards however specify multiple ways to quantify PDV.

RFC 5481 specifies two forms of packet delay variation …

- Inter-Packet Delay Variation, IPDV, where the reference is the previous packet in the stream (according to sending sequence), and the reference changes for each packet in the stream. Properties of variation are coupled with packet sequence in this formulation. 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, 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 requires knowledge of time at source and destination. They are typically represented by histograms showing statistical distribution of delay variation. Note that packet loss has great influence for results in both cases (extreme cases are described in the RFC). For reporting and SLA purposes simplicity is important and PDV lends itself better (percentiles, median, mean, etc.).

PDV metrics can be used with different stream characteristics such as Poisson streams [RFC 3393] and Periodic streams [RFC 3432] depending on the purpose and testing environment.

Ixia does not fully implement RFC 5481, however it does provide a packet delay variation measurement (in 3 modes: FIFO, LILO, FILO). Ixia uses the absolute value of delay variation as formulated below.

- DV of packet 2 = ABS(Latency of Packet 2 – Latency of Packet 1)
- DV of packet 3 = ABS(Latency of Packet 3 – Latency of Packet 2)

Note: Sequence checking should be enabled (results obtained with out-of-order packets are not valid).

4.0 Test Setup

Figure 4-1 below shows the vE-CPE test setup used for collecting test results in this report.

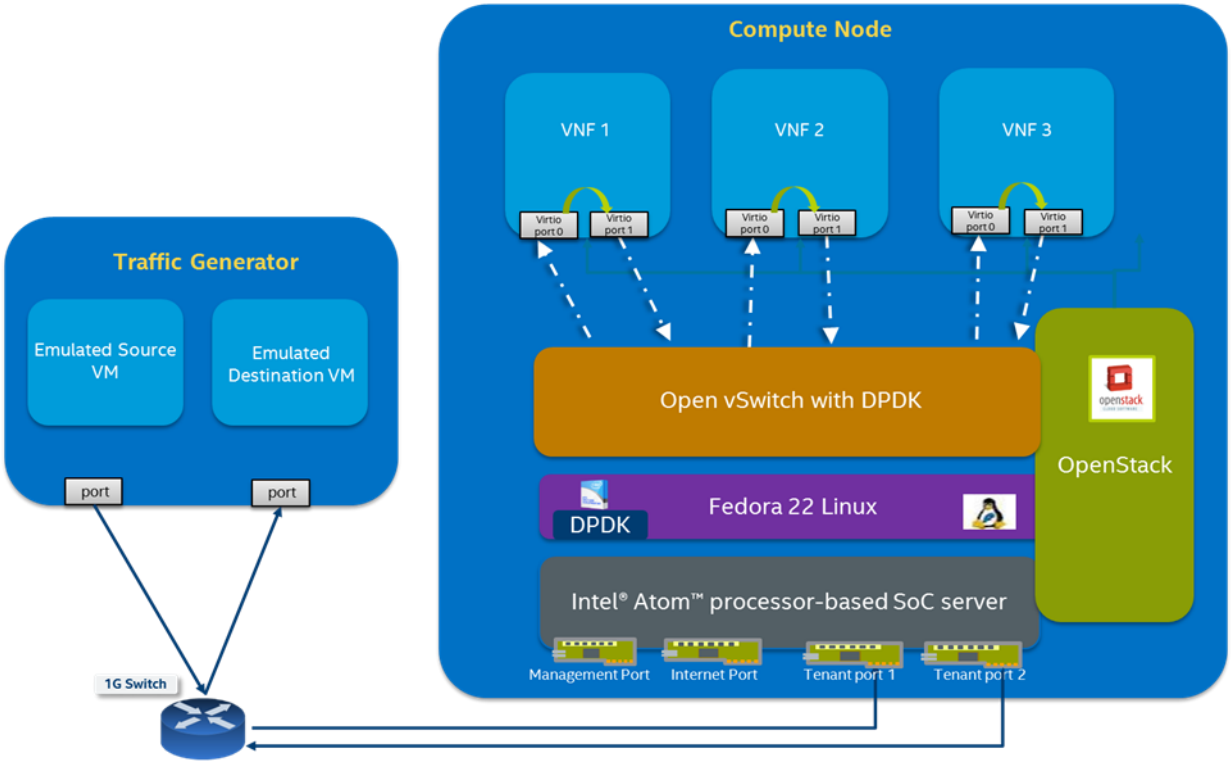

Figure 4-1 vE-CPE Test setup showing compute node running on Intel® Atom™ Processor based platform

In this report, Intel® does not attempt to emulate the real characteristics of the vE-CPE network services (e.g. firewall, WAN optimizer) but rather assume that there are 3 VNFs simply forwarding packets. Intel® Data Plane Performance Demonstrator (Intel® DPPD) code is used for this purpose. Intel® DPPD are Linux user space DPDK-based applications intended for investigation packet processing applications on Intel® platforms. See 01.org for more information on Intel® DPPD.

4.1 Hardware Components

Table 4-1 Intel® Atom™ Processor C2750 SoC platform - hardware ingredients

Item	Description	Notes
Platform	SuperMicro SuperServer 5018A-FTN4	Intel® Atom™ processor-based SoC server Motherboard: SuperMicro A1Sri-2758F 4 x 1GbE integrated Intel® Ethernet C2000 SoC I354 Quad GbE LAN ports, 120 GB SSD 2.5in SATA 6GB/s Intel® Wolfsville SSDSC2BB120G4
Processor	Intel® Atom™ processor C2758	8 core, 8 threads, 2.4 GHz, 4 MB cache CPU TDP 20W (8-Core) FCBGA 1283 System-on-Chip
Memory	32 GB 1600MHZ DDR3L ECC CL11 SODIMM 1.35V	4x 204-pin DDR3 SO-DIMM slots Supports up to 64GB DDR3 ECC memory
BIOS	AMIBIOS Version: 1.1 Release Date: 01/09/2015	Hyper-Threading not applicable Intel® Virtualization Technology (Intel® VT-x) enabled
Network Interfaces	4x Integrated Gigabit Ethernet controller (1 GbE)	The four GbE interfaces on the platform can be configured as a 2.5 GbE interface to add up to 10 GbE interface. However 2.5 GbE is not a standard interface (although used in backplane configurations) and using a 10 GbE NIC is recommended for 10 GbE purposes. We are not using a 10 GbE NIC for the vE-CPE use-case.
Accelerator options	QAT	QAT acceleration was not used for benchmarking purposes.
Security options	UEFI Secure Boot	

4.2 Software Components

The Table 4-2 describes functions of the software ingredients along with their version or configuration. For open-source components, a specific commit ID set is used for this integration. Note that the commit IDs detailed in the table are used as they are the latest working set at the time of this release.

Table 4-2 Software Versions

Software Component	Function	Version/Configuration
Fedora 22	Host Operating System	Fedora 22 Server x86_64 Kernel version: 4.1.10-200.fc22.x86_64
QEMU-KVM	Virtualization technology	QEMU-KVM version: 2.3.1-7.fc22.x86_64 libvirt version: 1.2.13.1-3.fc22.x86_64
DPDK	Network stack bypass and libraries for packet processing; includes user space vhost drivers	2.1.0
Open vSwitch	vSwitch	Open vSwitch 2.4.9 Commit ID 88058f19ed9aadb1b22d26d93e46b3fd5eb1ad32 used for Open vSwitch 2.4.9 (non-DPDK nodes) Open vSwitch with DPDK
OpenStack	SDN orchestrator	OpenStack Liberty Release (2015-10-15)
DevStack	Tool for OpenStack deployment	https://github.com/openstack-dev/devstack.git commit: 6ff54a3936d3b1999d625019889c2e5894be396d
DPPD (Intel® Data Plane Performance Demonstrator)	DPDK based application	DPPD Version 17
Intel® Ethernet Drivers	Ethernet drivers	Driver Version: igb-5.2.15-k

Table 4-2 Commit IDs for Major OpenStack Components

OpenStack Component	Commit ID, Tag, or Branch
OpenStack Nova	stable/liberty 6df6ad3ff32f2b1fe2978df1032002548ad8eb66
OpenStack Neutron	stable/liberty 6dcfe3a9362ae5fcf18e5cfb59663e43446cd59c
OpenStack Keystone	stable/liberty 8dcd82fb9c76d43f26338bee293b32f4af473ad9
OpenStack Glance	stable/liberty 69516fad5f651a085a047a337a05c58b39023c1b
OpenStack Horizon	stable/liberty 593f0b78eea8efbb6d833d66acc7ab4dc852159b
OpenStack Cinder	stable/liberty 61026d4e4f2a58dd84ffb2e4e40ab99860b9316a
OpenStack Requirements	stable/liberty 55c6058d302919fc6c435e9b7791fb8c5e680ba1
OpenStack Tempest	stable/liberty a1edb75d7901a9e338ab397d208a40c99c5fd9a1
OpenStack noVNC	stable/liberty 6a90803feb124791960e3962e328aa3cfb729aeb
OpenStack networking-odl	stable/liberty fec09899a610d565baadbf33713b57e760aa27a5
OpenStack networking-ovs-dpdk	6a8821c69b9154bc432f86c62e478d14f1c6fcb5

Note: See Intel® ONP Release 2.0 Scripts available on 01.org for commit IDs of minor components.

5.0 Test Results

Table 5-1 below is a summary of the test cases and their configuration differences. Host performance provides a baseline of bare-metal packet forwarding using 2x1 GbE ports and 2 physical cores to forward packets with DPDK. The virtual switching test case compares performance of native OvS and OvS with DPDK. All the test cases with either one VM or multiple VMs use Intel® ONP platform and OpenStack is used for provisioning the compute nodes and network configurations.

Table 5-1 Summary of test cases

Ref.	Test Description	Metrics	Packet Size (Bytes)	Test Duration	Flows per Port
Host Performance					
5.1	(PHY-PHY) Bare-metal 2 port configuration using 2 physical cores	Throughput Latency (min, max, avg)	64, 128, 256, 512, 1024, 1280, 1518	60 Seconds	1 bi-flow
vSwitch Performance					
5.2	(PHY-OVS-PHY) Native OvS	Throughput Latency (min, max, avg)	66, 128, 256, 512, 1024, 1280, 1518	60 Seconds	1 bi-flow
5.2	(PHY-OVS-PHY) OvS with DPDK	Throughput Latency (min, max, avg)	66, 128, 256, 512, 1024, 1280, 1518	60 Seconds	1 bi-flow
Intel® ONP with VMs					
5.3	1 VM (PHY-OVS-VM-OVS-PHY) OVS with DPDK using OpenStack for compute and network provisioning	Throughput Latency (min, max, avg)	70, 128, 256, 512, 1024, 1280, 1518	60 Seconds	1 uni-flow 1 bi-flow
5.4	2 VMs (PHY-OVS-VM-OVS-VM-OVS-PHY) OVS with DPDK using OpenStack for compute and network provisioning	Throughput Latency (min, max, avg)	70, 128, 256, 512, 1024, 1280, 1518	60 Seconds	1 uni-flow
5.5	3 VMs (PHY-OVS-VM-OVS-VM-OVS-VM-OVS-PHY) OVS with DPDK using OpenStack for compute and network provisioning	Throughput Latency (min, max, avg)	70, 128, 256, 512, 1024, 1280, 1518	60 Seconds	1 uni-flow 1 bi-flow 100 bi-flow 500 bi-flow 1000 bi-flow
5.5	3 VMs (PHY-OVS-VM-OVS-VM-OVS-VM-OVS-PHY)	Latency (min, max)	70	Over 1 hour	1 bi-flow

5.1 Host Performance

The test setup for measuring host (i.e. no vSwitch or VM) throughput performance is shown in Figure 5-1. This test creates a baseline for comparing more complex test cases as well as comparing "bare-metal" performance between different platforms. A reliable baseline is important when trying to establish "apples-for-apples" comparisons for more complex test scenarios between different platforms. "Bare-metal" performance should be used to calibrate performance between platforms on a per-core / per-port basis to help interpret more complex scenarios.

In this setup tests attempt to achieve system throughput of 2 Gbps, using a 2-port configuration using 1 physical core with 1 PMD thread (no hyper-threading). Two physical cores were assigned to the forwarding application. Tests use 2 ports with 1 bidirectional flow (i.e. 2 unidirectional flows through each port). Line-rate is 2 Gbps which is the maximum bidirectional throughput across each GbE port.

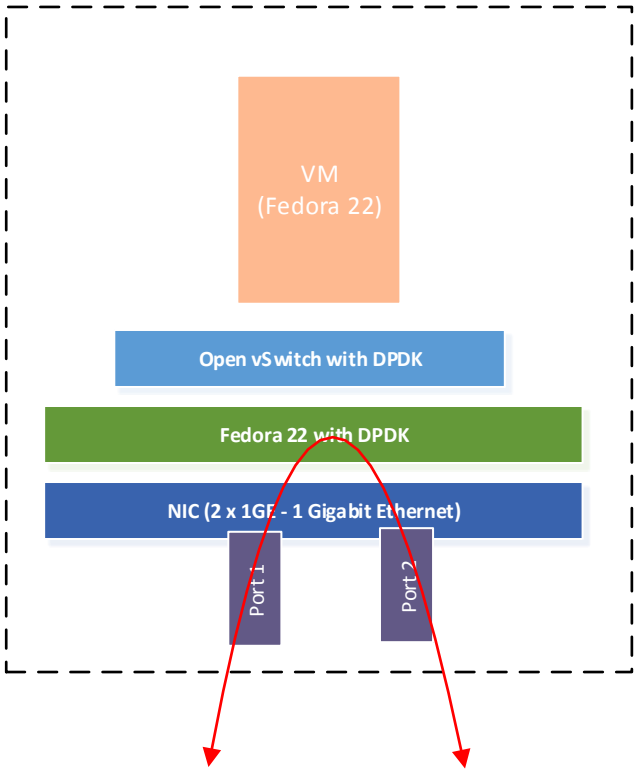

Figure 5-1 Host Performance test setup (PHY-PHY)

Test results are shown in the Table 5-2 for all packet sizes. Line-rate is achieved for all packet sizes. Average, minimum and maximum latency is consistent across packet sizes. Maximum latency is approximately 250 µs.

Table 5-2 PHY-PHY test results

	Bidirectional throughput with 0 frame loss					
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (µs)
66	2,000	2,906,970	100	151	32	269
128	2,000	1,689,185	100	143	10	276
256	2,000	905,795	100	160	60	261
512	2,000	469,924	100	158	62	254
1024	2,000	239,463	100	141	33	249
1280	2,000	192,307	100	134	22	247
1518	2,000	162,548	100	152	44	261
Affinity Details	1PMD thread and 0 frame loss resolution ./build/l2fwd -c 0x6 -n 2 -- -p 0x3 2P Onboard NIC					

5.2 Virtual Switching Performance

Virtual switching tests attempt to achieve aggregated system throughput of 2 Gbps (i.e. line-rate) through GbE ports and compares performance between the native OvS and OvS with DPDK. Each port handles 1 bidirectional flow for a total of 2 unidirectional flows. Figure 5-2 shows the test setup for PHY-OVS-PHY with two 1 GbE ports. The setup used 1 physical core with 1 PMD thread (no hyper-threading).

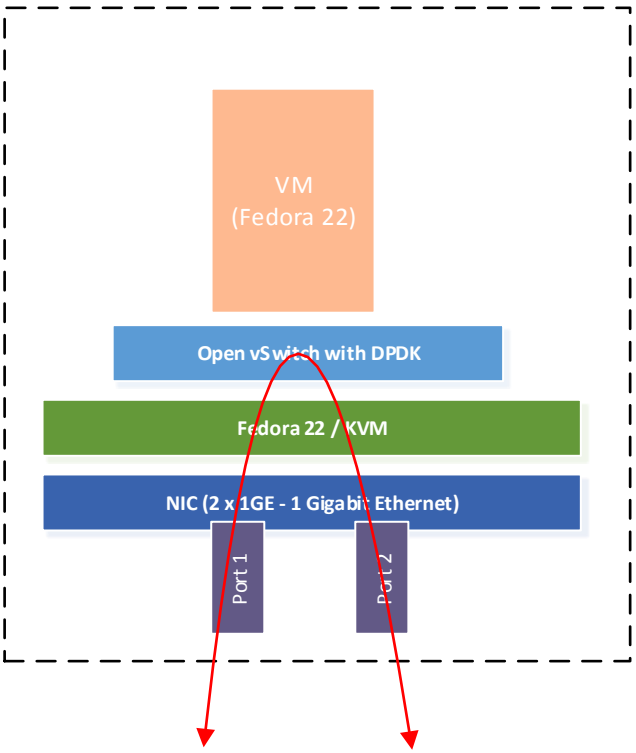

Figure 5-2 Virtual switching performance test setup (PHY-OVS-PHY)

Note that one switching operation takes place in the vSwitch while packets are being routed through the system.

Figure 5-3 shows throughput results comparing native OvS and OvS with DPDK as measured in Mbps. The same results are depicted in Figure 5-4 however this time shown as a percent of line-rate. The following observations are made:

- For packets over 1024 Bytes, both vSwitch configurations achieve line-rate
- OvS with DPDK achieves line-rate with all packets sizes over 128 Bytes
- For smallest packet sizes OvS with DPDK achieves 5x throughput of native OvS (64% of line-rate versus 13%)
- Latency measurements are not identical but are comparable.

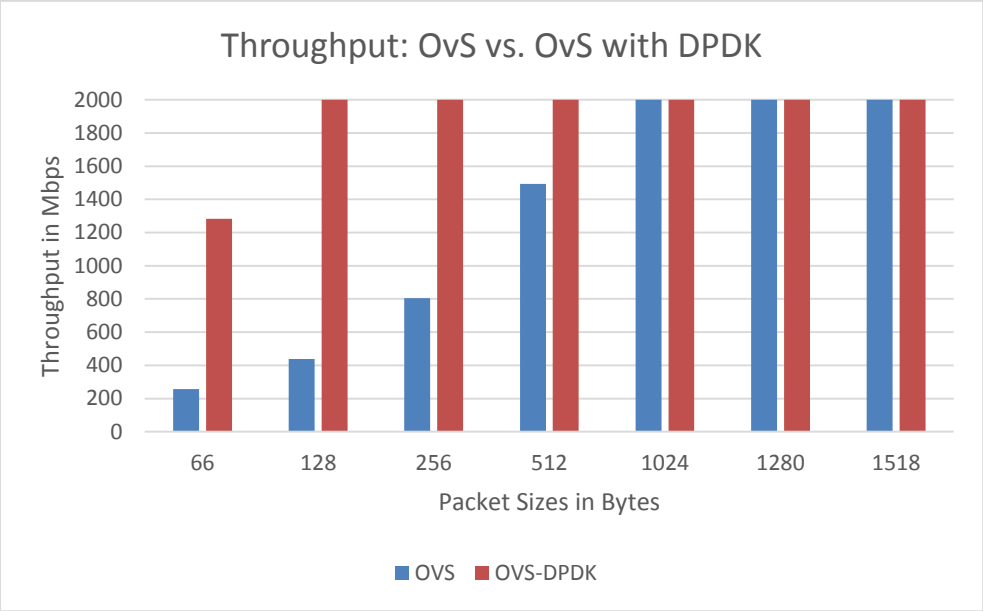

Figure 5-3 PHY-OVS-PHY test results - throughput performance of native OvS and OvS with DPDK

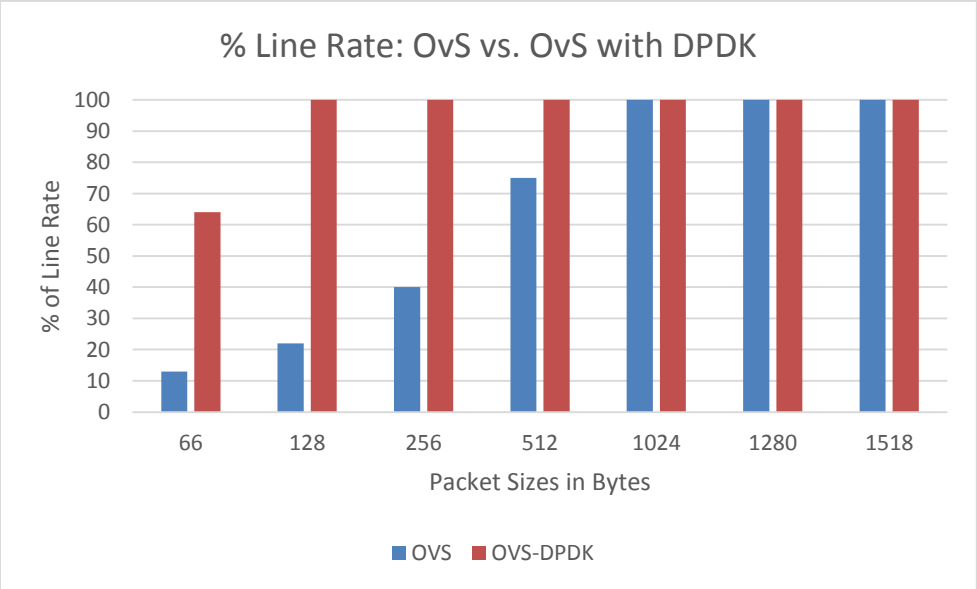

Figure 5-4 PHY-OVS-PHY test results - % line rate of native OvS and OvS with DPDK

Table 5-3 PHY-OVS-PHY test results – throughput for native OvS

Bidirectional throughput with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
66	256	372,457	13	121	5	237
128	439	370,830	22	136	5	267
256	805	364,442	40	117	6	228
512	1,494	350,975	75	175	11	339
1024	2,000	239,463	100	159	39	280
1280	2,000	192,307	100	161	56	266
1518	2,000	162,548	100	426	62	790
Affinity Details	2P Onboard NIC CPU Isolation					

Table 5-4 PHY-OVS-PHY test results – throughput for OvS with DPDK

Bidirectional Throughput with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
66	1,283	1,864,557	64	196	18	374
128	2,000	1,689,186	100	91	18	165
256	2,000	905,795.9	100	89	17	161
512	2,000	469,924	100	66	17	115
1024	2,000	239,463	100	65	17	113
1280	2,000	192,307	100	89	18	161
1518	2,000	162,548	100	66	18	114
Affinity Details	1PMD thread based OvS and 0 frame loss resolution # ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=4 # ./ovs-vsctl set Open_vSwitch . other_config:max-idle=30000 2P Onboard NIC					

5.3 Intel® ONP with 1 VM

One VM test configuration attempts to achieve aggregated system throughput of 2 Gbps (i.e. line-rate) through two 1 GbE ports and compares performance between one uni-directional flow and one bi-directional flow using OvS with DPDK. The test setup uses OvS with DPDK as depicted in Figure 5-5. OpenStack is used to provision the VM and internal network of the compute node. The VM is assigned two physical cores using the dedicated CPU policy of OpenStack flavor's configuration parameter; and is configured as a L2 forwarding VNF using DPPD. The OvS PMD and OvS vSwitchd threads are pinned to one physical core each using the options provided by networking-ovs-dpdk ML2 plugin.

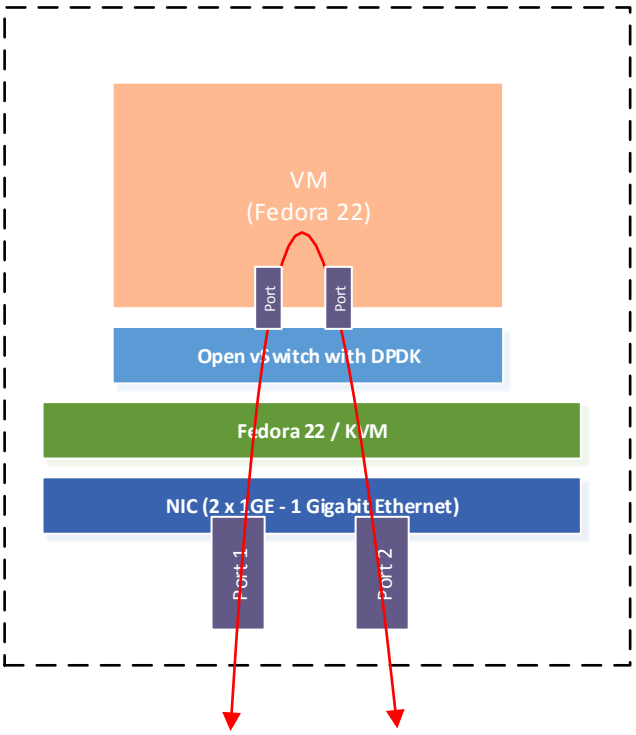

Figure 5-5 One VM performance test setup (PHY-OVS-VM-OVS-PHY)

Note that two switching operations take place in the vSwitch while packets are being routed through the system.

The graph in Figure 5-6 below compares the maximum throughput achieved with a unidirectional flow versus a bidirectional flow measured in Mbps. Figure 5-7 shows the same data as % of line-rate. Note that for a unidirectional flow line-rate is 1 Gbps whereas for a bidirectional flow line-rate is 2 Gbps. The following observations can be made:

- For smallest packets bi-directional throughput is double that of uni-directional throughput as would ideally be expected.
- For larger packet sizes bi-directional throughput is more than double that of uni-directional throughput which requires further analysis to understand.
- Both uni-directional and bi-directional flows approach line-rate for packet sizes greater than 1024 Bytes.

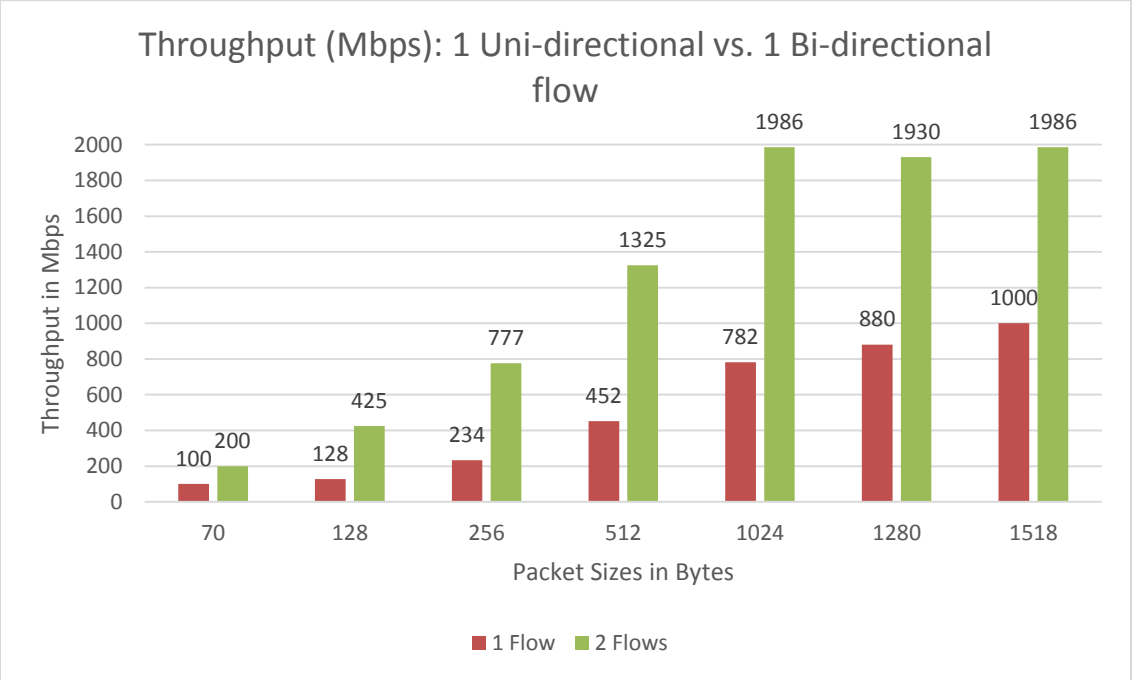

Figure 5-6 One VM test results – OvS with DPDK throughput with 1 unidirectional and 1 bidirectional flows per port

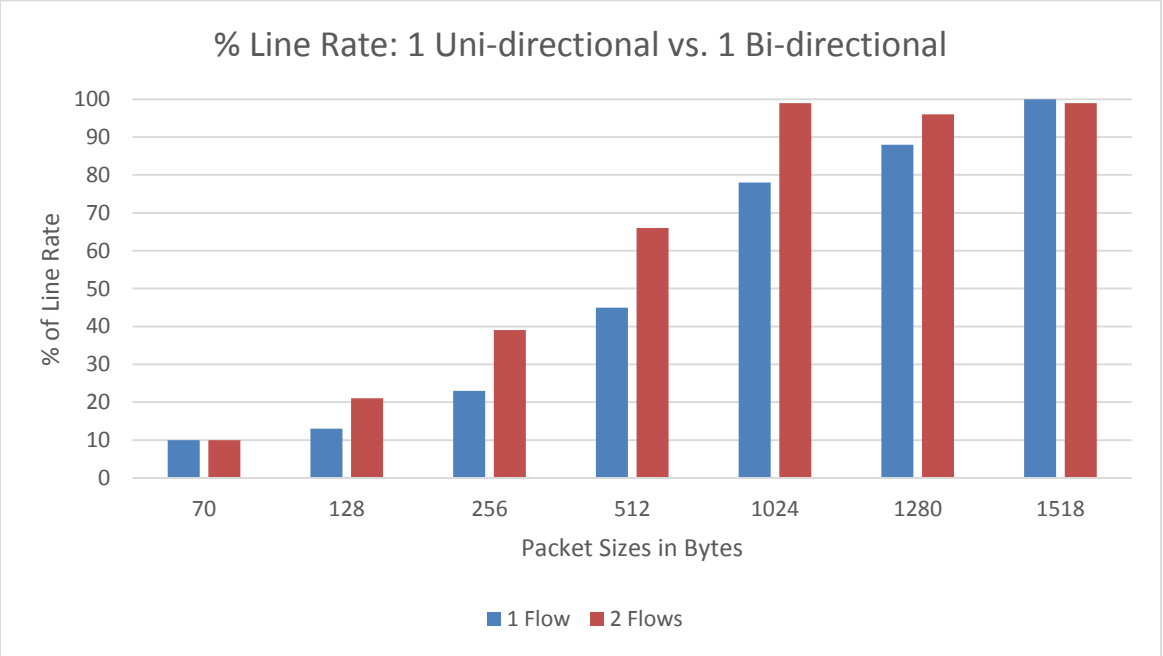

Figure 5-7 One VM test results – % line rate for OvS with DPDK with 1 unidirectional and 1 bidirectional flows per port

Table 5-5 1VM test results for OvS with DPDK with 1 unidirectional flow

Unidirectional throughput with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	100	138,882	10	2550	7	5093
128	128	108,214	13	732	29	1434
256	234	105,795	23	740	32	1449
512	452	106,100	45	747	40	1454
1024	782	93,634	78	775	41	1508
1280	880	84,661	88	763	41	1484
1518	1000	81,272	100	822	139	1506
Affinity Details	1PMD thread based OvS and 0 frame loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard NIC DPPD running handle_l2fwd.cfg					

Table 5-6 1VM test results for OvS with DPDK with 1 bidirectional flow

Bidirectional throughput with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	200	277,771	10	1,610	8	3,213
128	425	358,954	21	1,498	9	2,987
256	777	351,705	39	574	10	1,139
512	1,325	311,326	66	575	9	1,141
1024	1,986	237,780	99	2,470	31	4,908
1280	1,930	185,547	96	2,430	11	4,850
1518	1,986	161,400	99	2,619	24	5,213
Affinity Details	1PMD thread based OvS and 0 frame loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard NIC DPPD running handle_l2fwd.cfg					

5.4 Intel® ONP with 2 VMs

Figure 5-8 shows the VM-VM test setup with 2 x 1GbE ports. Two VM test configuration attempts to achieve aggregated system throughput of 2 Gbps (i.e. line-rate) through two 1GbE ports and compares performance between one uni-directional flow and one bi-directional flow using OvS with DPDK. The test setup uses OvS with DPDK as depicted in Figure 5-8. OpenStack is used to provision the VMs and internal network of the compute node. The VMs are assigned two physical cores each using the dedicated CPU policy of OpenStack flavor's configuration parameter; and are configured as L2 forwarding VNFs using DPPD. The OvS PMD and OvS vSwitchd threads are pinned to one physical core each using the options provided by networking-ovs-dpdk ML2 plugin.

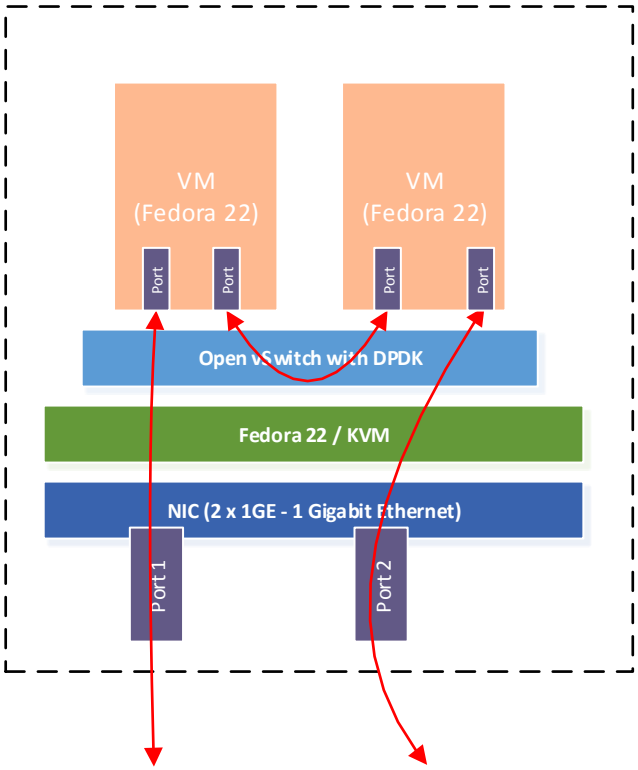

Figure 5-8 Two VMs performance test setup (PHY-OVS-VM-OVS-VM-OVS-PHY)

Note that there are 6 switching operations taking place in the vSwitch for bi-directional traffic (i.e. 3 switching operations for uni-directional traffic).

Table 5-7 and

Table 5-8 below provide data for maximum throughput achieved with a unidirectional flow and a bidirectional flow respectively. Note that for a unidirectional flow line-rate is 1 Gbps whereas for a bidirectional flow line-rate is 2 Gbps. The following observations can be made:

- Throughput for two VMs are comparable with one VM
- For smallest packets bi-directional throughput is almost double that of uni-directional throughput as would ideally be expected.
- For larger packet sizes bi-directional throughput is more than double that of uni-directional throughput which requires further analysis to understand.

Table 5-7 Two VMs test results with 1 unidirectional flow

Unidirectional throughput with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	100	138,880	10	766	9	1,524
128	100	84,460	10	760	9	1,511
256	177	80,319	18	1,053	90	2,015
512	339	79,667	34	989	118	1,860
1024	634	75,955	63	1,063	177	1,949
1280	782	75,195	78	1,313	203	2,423
1518	895	72,702	89	3,515	264	6,766
Affinity Details	1PMD thread based OvS and 0 frame Loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard NIC DPPD running handle_l2fwd.cfg					

Table 5-8 Two VMs test results with 1 bidirectional flow

Bidirectional throughput with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	175	242,622	9	617	11	1,223
128	298	252,060	15	627	13	1,242
256	546	247,255	27	632	15	1,249
512	190	44,680	10	1,307	12	2,602
1024	1,768	211,683	88	2,213	23	4,402
1280	1,490	143,225	74	947	14	1,880
1518	1,675	136,148	84	1,735	15	3,454
Affinity Details	1PMD thread based OvS and 0 frame loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard NIC DPPD running handle_l2fwd.cfg					

5.5 vE-CPE – Intel® ONP with 3 VMs

Figure 5-9 shows the three VM test setup with 2 x 1 GbE ports. Three VM test configuration attempts to achieve aggregated system throughput of 2 Gbps (i.e. line-rate) through two 1GbE ports and compares performance between one uni-directional flow and various bi-directional flows (1, 100, 500, 1000)using OvS with DPDK. OpenStack is used to provision the VMs and internal network of the compute node. The VMs are assigned two physical cores each using the dedicated CPU policy of Open Stack flavor's configuration parameter; and are configured as L2 forwarding VNFs using DPPD. The OvS PMD and OvS vSwitchd threads are pinned to one physical core each using the options provided by networking-ovs-dpdk ML2 plugin. More configuration details

are provided in 8.0 vE-CPE Setup with Intel® ONP and performance fine tuning methods are discussed in 9.0 Performance Tuning . Other than throughput, minimum and maximum latency of a single bidirectional flow with 1 hour test duration was measured.

Figure 5-9 shows the test setup with 2 x 1 GbE ports with packets being forwarded from the first VM to the second VM, then to the third VM (in the case of a unidirectional flow). Note that there are 8 switching operations taking place while packets are being routed through the system for one bidirectional flow.

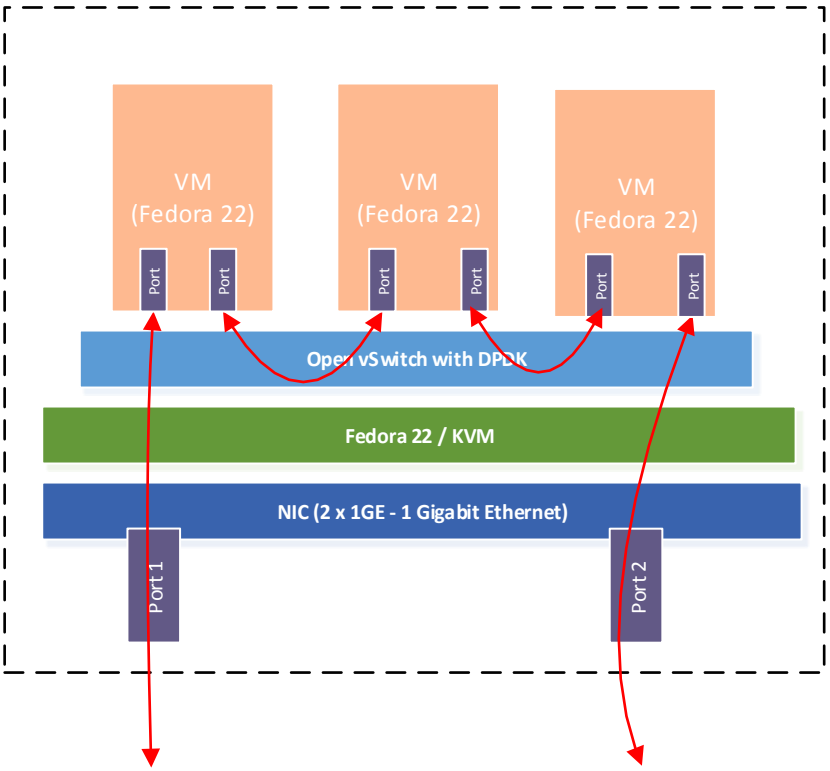

Figure 5-9 Three VMs performance test setup (PHY-OVS-VM-OVS-VM-OVS-VM-OVS-PHY)

Discussion of the results below is provided in section 0.

5.5.1 Intel® ONP with 3 VMs – 1 Unidirectional Flow

Table 5-9 Three VMs test results with 1 unidirectional flow

Unidirectional throughput with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	103	143,008	10	19,597	263	38,931
128	165	139,320	16	19,214	256	38,172
256	295	133,563	29	19,941	263	39,619
512	536	126,028	54	19,967	268	39,667
1024	941	112,685	94	22,877	293	45,462
1280	997	95,912	100	33,525	283	66,767
1518	996	80,910	100	36,787	278	73,295
Affinity Details	1PMD thread based OvS and 0 Frame Loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard NIC DPPD running handle_l2fwd.cfg					

5.5.2 Intel® ONP with 3 VMs – 1 Bidirectional Flow

Table 5-10 Three VMs test results with 1 bidirectional flow

Bidirectional throughput with 0 frame Loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	20	27,778	1	607	15	1,200
128	20	16,879	1	29,766	16	59,516
256	66	30,075	3	553	16	1,090
512	840	197,332	42	717	66	1,368
1024	1,505	180,197	75	994	118	1,870
1280	639	61,419	32	2,123	18	4,229
1518	376	30,542	19	2,501	19	4,983
Affinity Details	1PMD thread based OvS and 0 Frame Loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard NIC DPPD running handle_l2fwd.cfg					

5.5.3 Intel® ONP with 3 VMs – 100 Bidirectional Flows

Table 5-11 Three VMs test results with 100 bidirectional flows

Bidirectional throughput with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	20	27,778	1	1,493	29	2,958
128	144	12,1411	7	1,284	99	2,468
256	66	30,075	3	1,641	30	3,253
512	283	66,487	14	1,114	49	2,179
1024	144	17,212	7	1,438	33	2,843
1280	206	19,772	10	6,243	34	12,452
1518	561	45,628	28	799	34	1,563
Affinity Details	1PMD thread based OvS and 0 frame loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard NIC DPPD running handle_l2fwd.cfg					

5.5.4 Intel® ONP with 3 VMs – 500 Bidirectional Flows

Table 5-12 Three VMs test results with 500 bidirectional flows

Bidirectional throughput with 0 facket loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	20	27,778	1	6,237	28	12,445
128	144	121,411	7	916	110	1,722
256	175	79,116	9	736	53	1,418
512	268	62,853	13	816	45	1,588
1024	530	63,514	27	1,536	58	3,014
1280	1,010	97,116	51	1,146	170	2,122
1518	221	17,969	11	1,846	34	3,657
Affinity Details	1PMD thread based OvS and 0 frame loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard NIC DPPD running handle_l2fwd.cfg					

5.5.5 Intel® ONP with 3 VMs – 1000 Bidirectional Flows

Table 5-13 Three VMs test results with 1000 bidirectional flows

Bidirectional Throughput with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	20	27,756	1	10,363	29	20,698
128	128	108,346	6	827	70	1,583
256	175	79,116	9	810	51	1,569
512	144	33,776	7	725	31	1,419
1024	886	106,113	44	1,672	211	3,134
1280	546	52,494	27	1,973	34	3,912
1518	190	15,455	10	7,949	35	15,863
Affinity Details	1PMD thread based OVS and 0 frame loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard Nic DPPD running handle_l2fwd.cfg					

5.5.6 Intel® ONP with 3 VMs – Latency

Latency for a single bidirectional flow with 1 hour test duration was measured.

Table 5-14 Three VMs test results – latency with 1 bidirectional flow

Bidirectional flow with 0 frame loss						
Packet Size	Mbps	Packets/sec	% Line Rate	Average Latency (us)	Minimum Latency (us)	Maximum Latency (us)
70	20	27,761	1	29,512	15	59,009
Affinity Details	1PMD thread based OvS and 0 frame loss resolution OVS_CORE_MASK=0x02 OVS_PMD_CORE_MASK=0x04 2P Onboard NIC DPPD running handle_l2fwd.cfg					

5.6 Comparison of Intel® ONP VM Test Results

This section provides a comparison of system performance using 1, 2, and 3 VMs for unidirectional and bidirectional flows. Note that all data represents zero frame loss for throughput tests. Maximum theoretical throughput for unidirectional flow is 1 Gbps and for bidirectional flows is 2 Gbps. Using OpenStack the following core allocation policies were configured:

- 1 physical core allocated to PMD across 2 physical ports
- 1 physical core allocated to vswitchd daemon
- 2 physical cores allocated per VM instance (i.e. for 3 VMs 6 cores are dedicated)

Note that due to a bug with stable\Liberty (OpenStack) core-isolation could not be enabled in the grub menu. This means that while cores are allocated to various processes as above, there is no guarantee that they are being exclusively used by the corresponding process.

5.6.1 3 VMs – Unidirectional vs. Bidirectional Flows

The following observations can be made:

- Unidirectional flow achieves line rate for packets great than 1024 Bytes
- Bidirectional flow does not achieve line rate and throughput decreases for packets larger than 1024 Bytes. This is unexpected and requires investigation to understand.

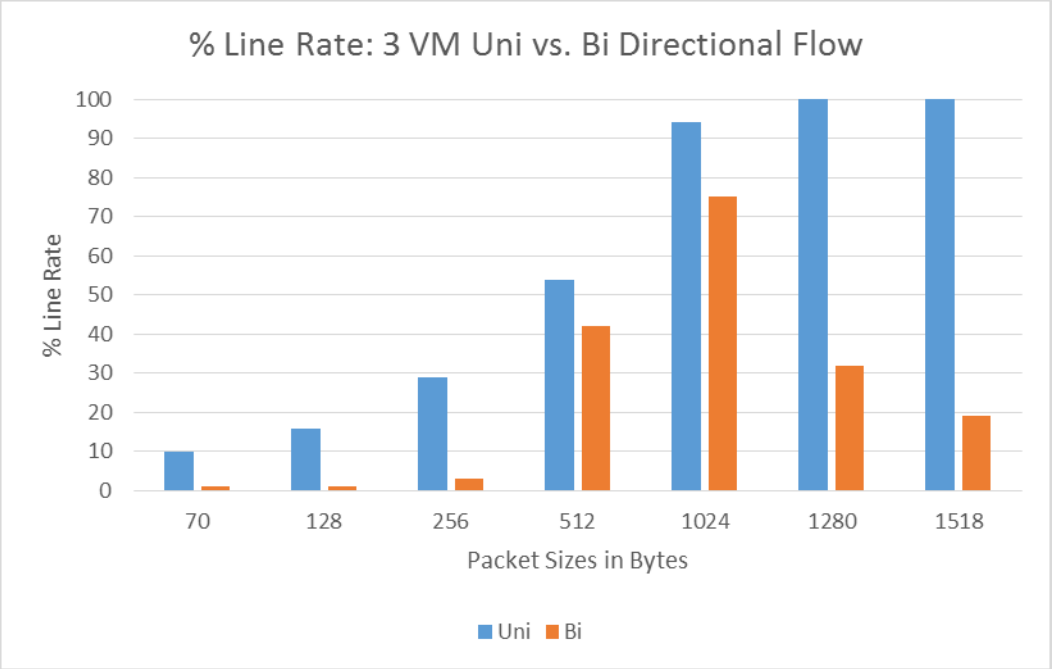

Figure 5-10 3 VMs % line rate comparison - unidirectional vs. bidirectional flows

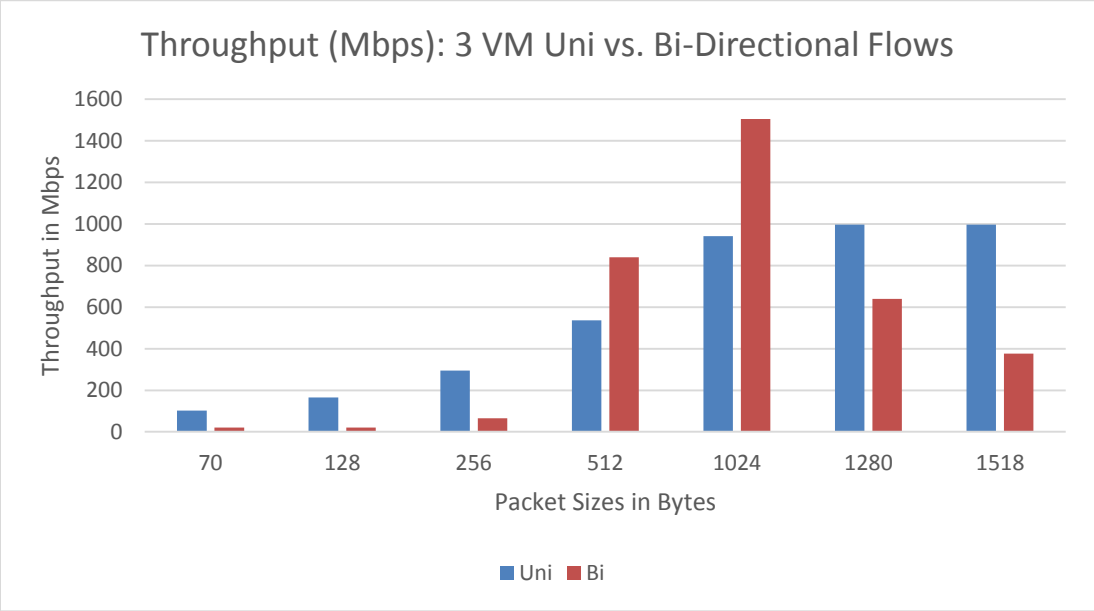

Figure 5-11 3 VMs throughput comparison – unidirectional flow vs. bidirectional flow

5.6.2 3 VMs – Comparison of Multiple Bidirectional Flows

The following observations can be made:

- For smallest packet size throughput is the same for all cases and is approximately 1% of line-rate (i.e. 20 Mbps).
- Ideally one flow would be expected to have the highest throughput however there are anomalies that require further investigation e.g. for packet size of 1518 Bytes, 100 flows have higher throughput than one flow.

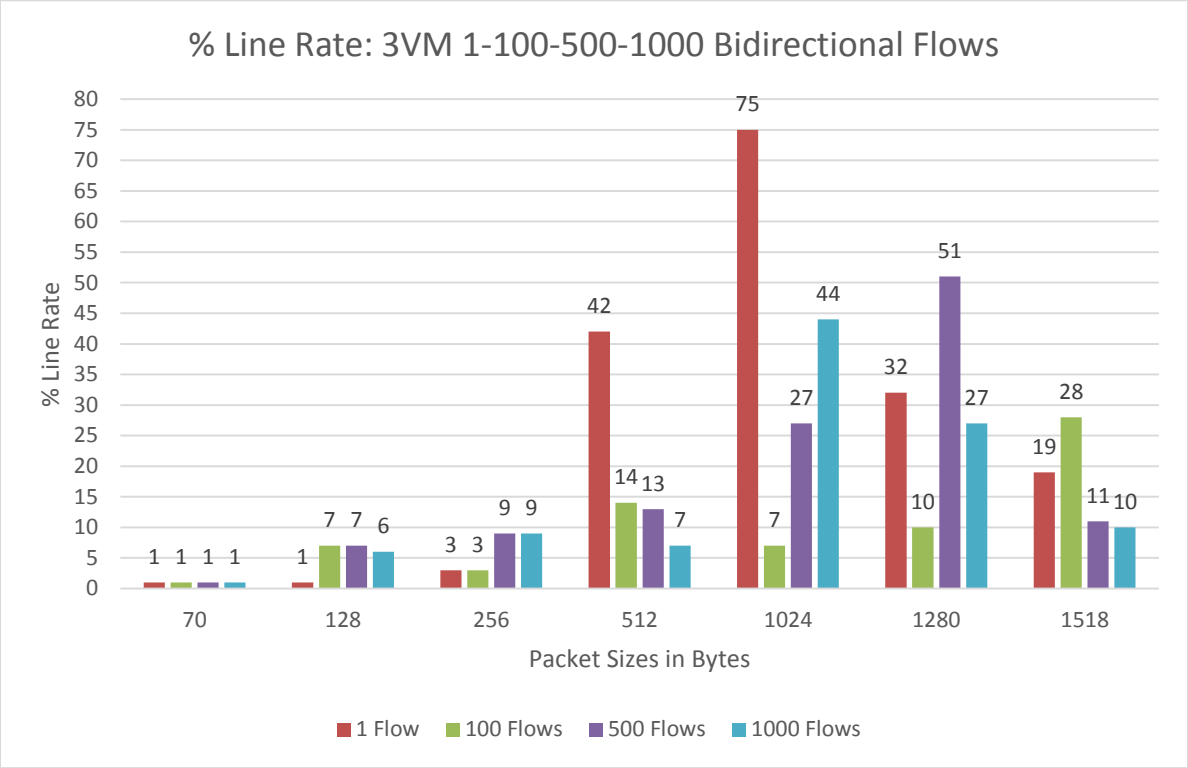

Figure 5-12 3 VMs % line rate comparison with various number of bidirectional flows

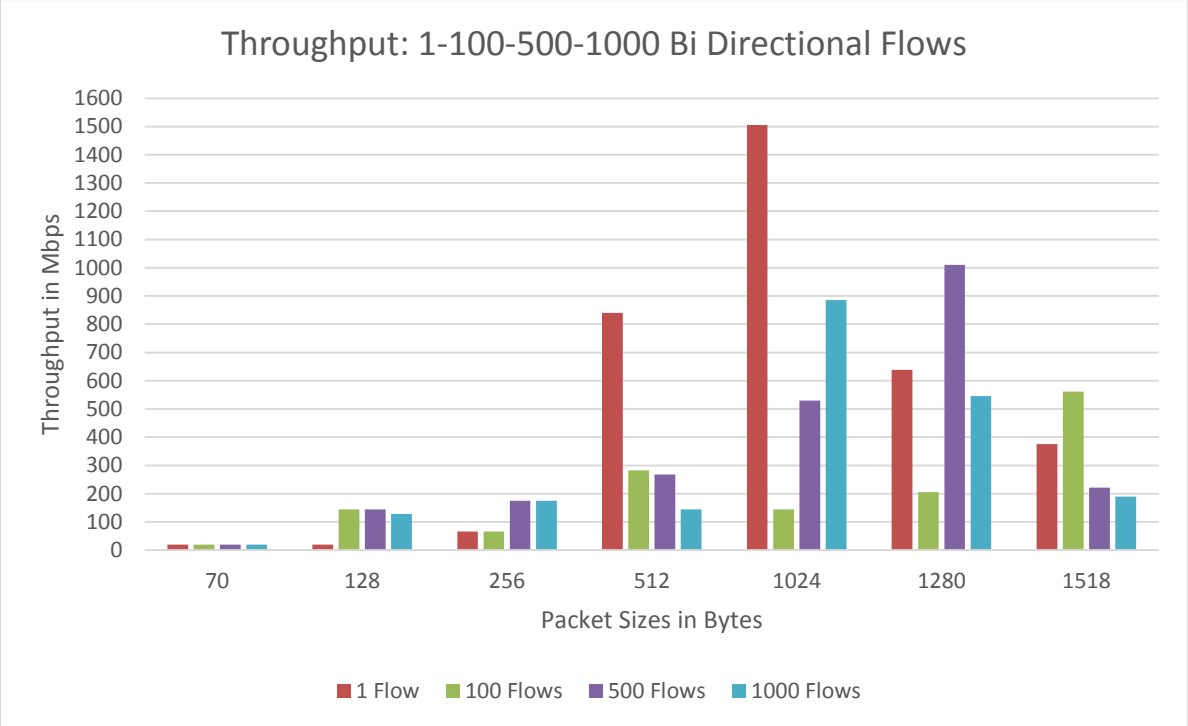

Figure 5-13 3 VMs throughput comparison with various number of bidirectional flows

5.6.3 Comparing Unidirectional Throughput – 1, 2, 3 VMs

The following observations can be made:

- Throughput with 2 VMs is lower than with both one VM and 3 VMs. This is unexpected and requires further investigation.
- The throughput scales as expected with a unidirectional flow for all configurations and packet sizes.

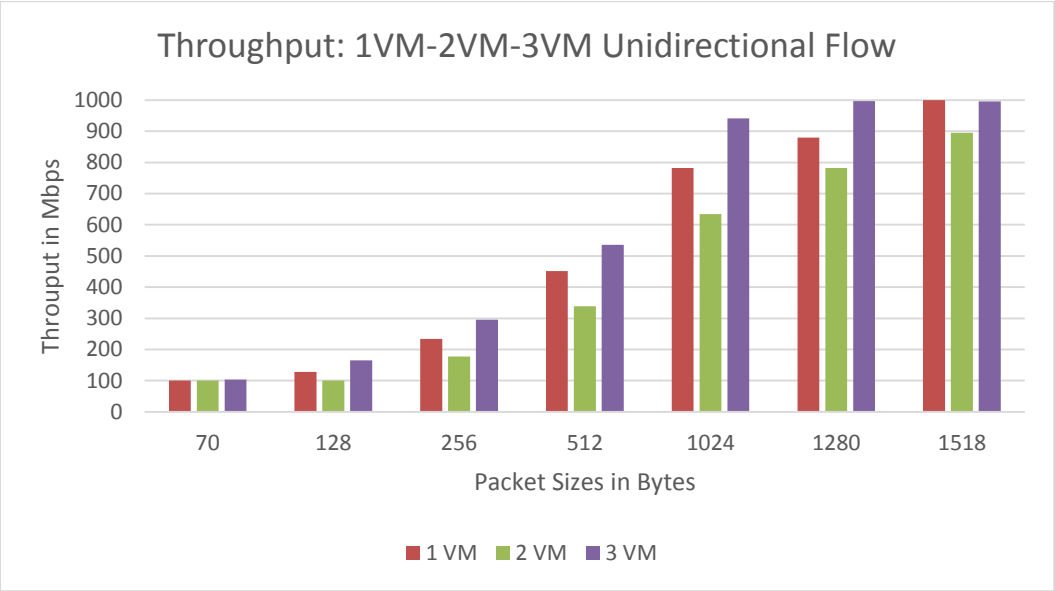

Figure 5-14 Throughput comparison for unidirectional flow and various number of VMs

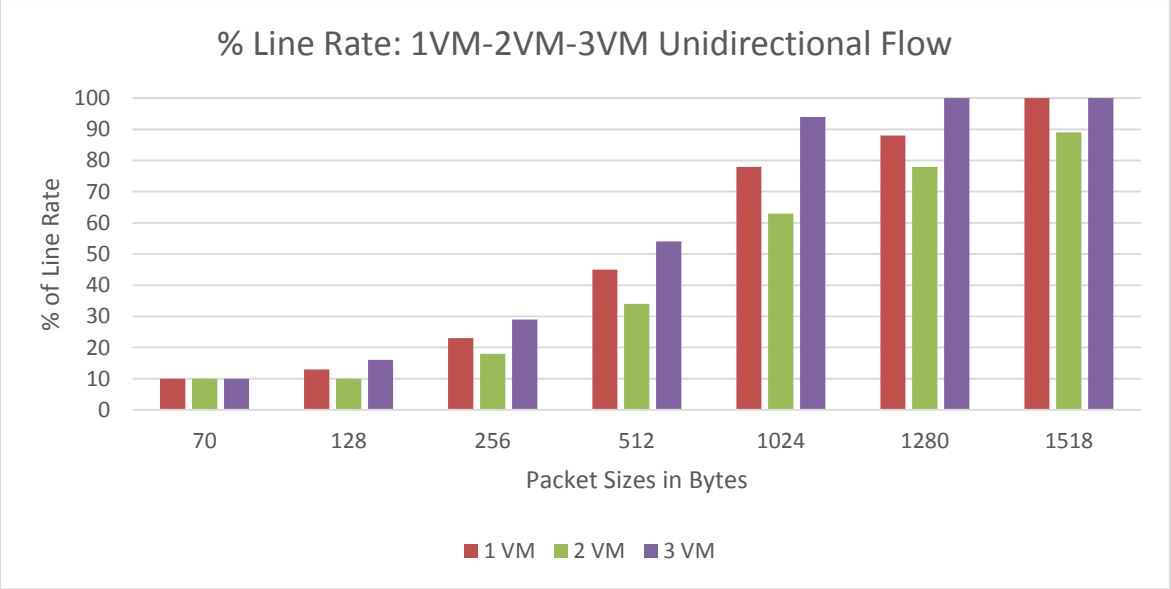

Figure 5-15 % line rate comparison for unidirectional flow and various number of VMs

5.6.4 Comparing Unidirectional Flow Latency – 1, 2, 3, VMs

The following observations can be made:

- Latency is similar for one and two VMs
- Latency for three VMs is significantly greater than for one and two VMs.

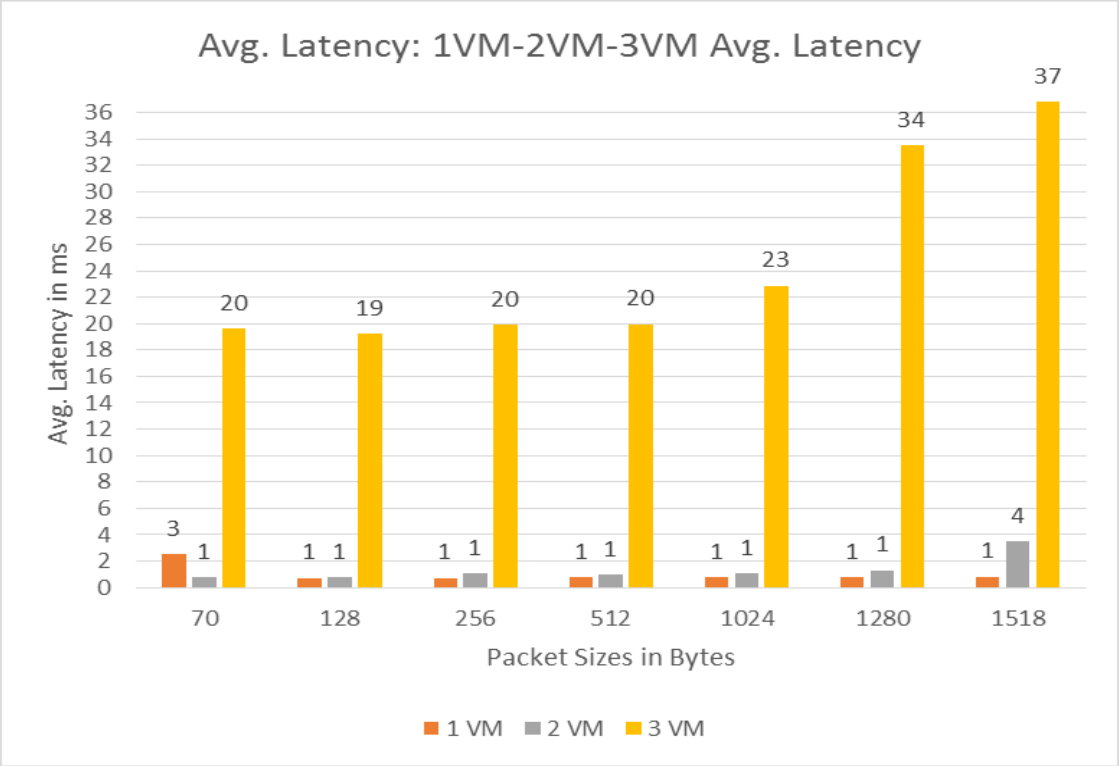

Figure 5-16 Average latency comparison for unidirectional flow and various number of VMs

Note that average latency data was collected over 60s for each test.

5.6.5 Comparing Bidirectional Throughput - 1, 2, 3 VMs

The following observations can be made:

- The throughput scales as expected for one VM with all packet sizes.
- Some anomalies exist for two and three VM case that need to be investigated to understand behavior.

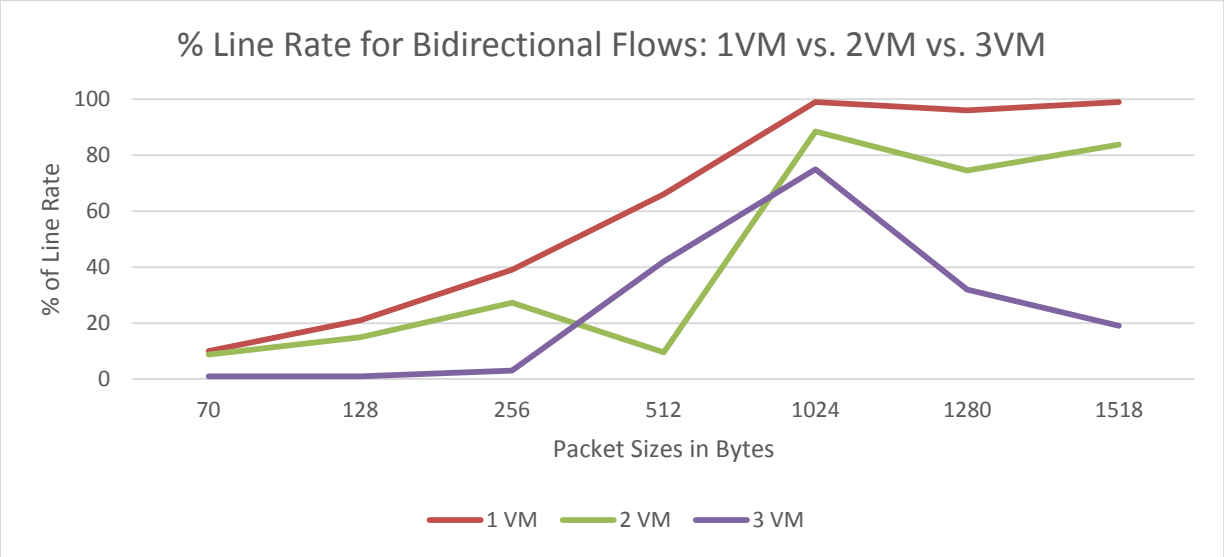

Figure 5-17 % line rate comparison for bidirectional flows and various number of VMs

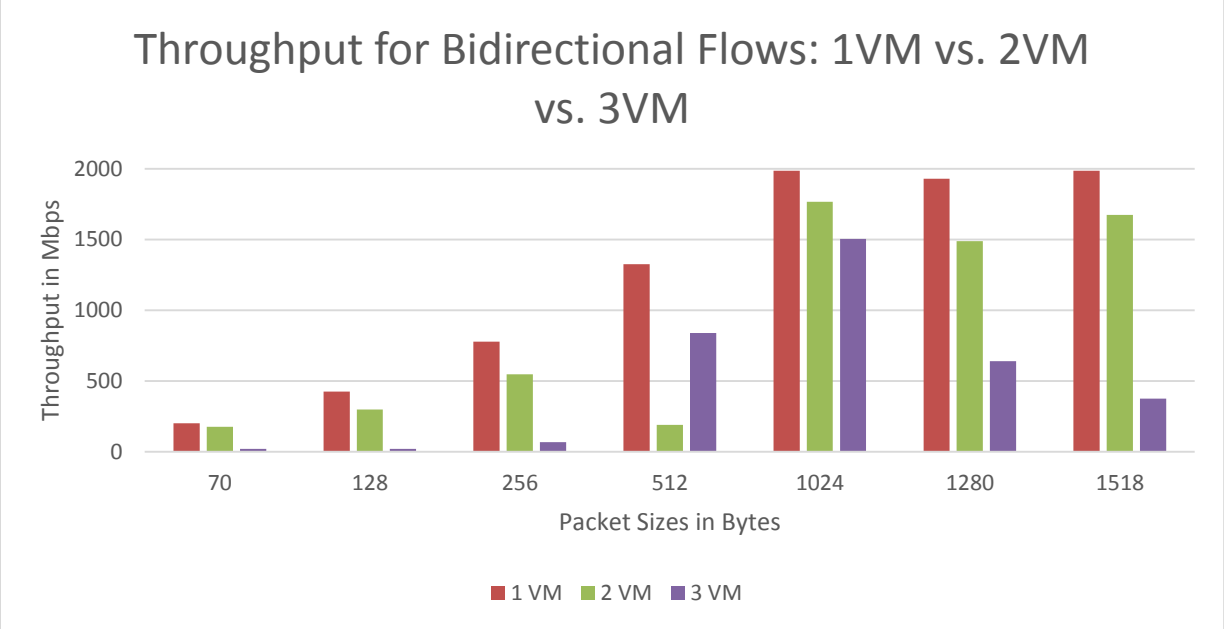

Figure 5-18 Throughput comparison for bidirectional flows and various number of VMs

5.6.6 Comparing Bidirectional Flow Latency – 1, 2, 3, VMs

The following observations can be made:

- Latency is similar for all configurations and packet sizes
- An anomaly is 128 Bytes with 3 VMs which has significantly larger latency than all other measurements.

Note that average latency data was collected over 60s for each test. This should be increased to at least one hour to observe how outliers could be affecting this data.

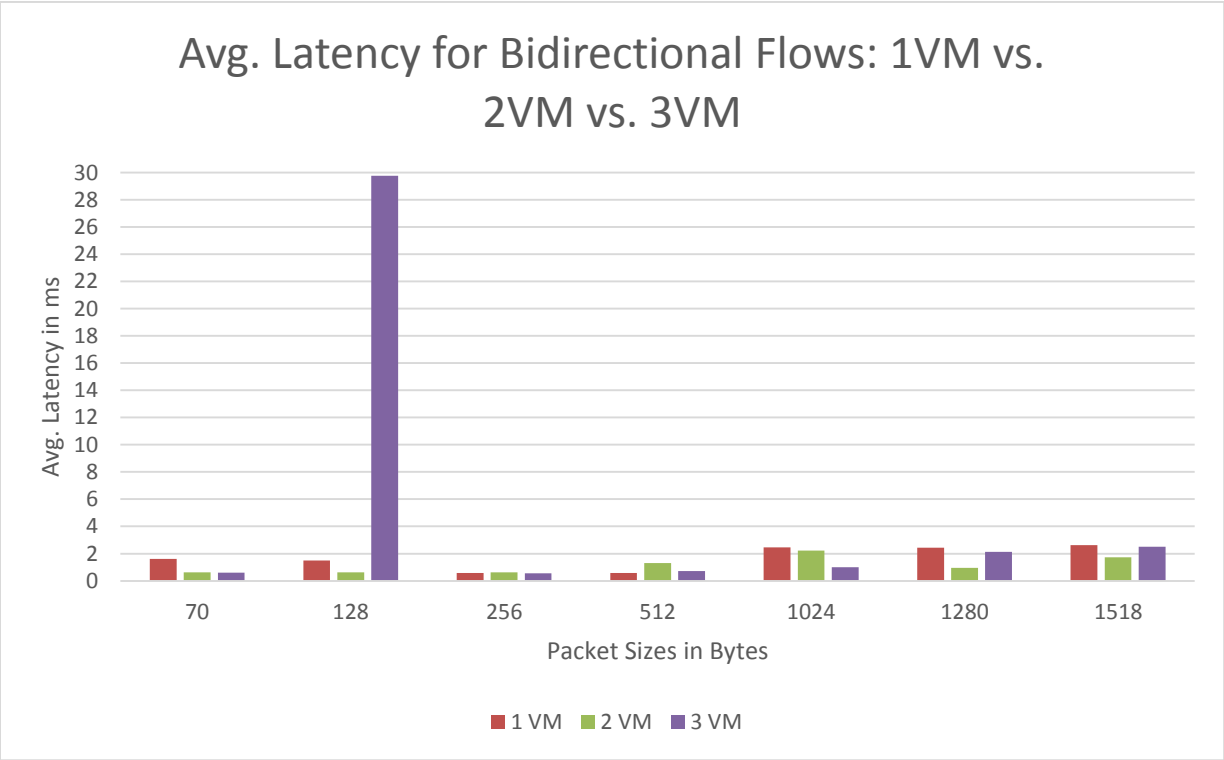

Figure 5-19 Average latency comparison for bidirectional flows and various number of VMs

6.0 PHY-PHY DPDK Host Performance Test Setup

This section assumes that Fedora 22 is already installed using the steps detailed in Intel® ONP Reference Architecture Guide.

6.1 Configure the Host Machine

1. Disable the interruption request (IRQ) balance:

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

2. Disable Firewall and iptables:

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

3. Disable Security-enhanced Linux (SELinux) by editing the file `/etc/selinux/config` as below:

```
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
# rmmod ipmi_ssif
```

6.2 Set the Kernel Boot Parameters

1. Add the following to the kernel boot parameters `/etc/default/grub` on the DUT to enable HugePages and isolate CPU cores 1-7:

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=fedora-server/root rd.lvm.lv=fedora-server/swap
default_hugepagesz=2M 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 2MB HugePage sizes are created. You should see 2048 2MB HugePages:

```
# ls /sys/devices/system/node/node0/hugepages/hugepages-*
hugepages-2048kB/
# cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
2048
```

6.3 Compile DPDK 2.1.0 with Intel® ONP 2.0 Commit ID

1. Download the DPDK 2.1.0 software from dpdk.org

```
# git clone http://dpdk.org/git/dpdk
# git reset --hard 7173acefc7cfdfbbb9b91fcba1c9a67adb4c07c9
```

2. Go to the dpdk directory and run the following:

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

3. Edit the `.config` file and set the configuration options:

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

4. Save the file and run make:

```
# EXTRA_CFLAGS="-g -Ofast"
# make -j8
```

6.4 Prepare to instantiate DPDK

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

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

2. Check that HugePages are mounted:

```
# mount
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 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 1 GbE NIC ports:

```
# lspci | grep Ethernet
00:14.0 Ethernet controller: Intel Corporation Ethernet Controller for 1GbE
```

```
00:14.1 Ethernet controller: Intel Corporation Ethernet Controller for 1GbE
00:14.2 Ethernet controller: Intel Corporation Ethernet Controller for 1GbE
00:14.3 Ethernet controller: Intel Corporation Ethernet Controller for 1GbE
```

6.5 Bind the 1 GbE NIC Ports to igb_uio Driver

1. To create a 2-port configuration:

```
# python $DPDK_DIR/tools/dpdk_nic_bind.py --bind=igb_uio 14:00.0
# python $DPDK_DIR/tools/dpdk_nic_bind.py --bind=igb_uio 14:00.1
# python $DPDK_DIR/tools/dpdk_nic_bind.py -status

Network devices using the DPDK-compatible driver:
============================================
0000:14:00.0 'Ethernet Controller for 1GbE +' drv=igb_uio unused=
0000:14:00.1 'Ethernet Controller for 1GbE +' drv=igb_uio unused=
Network devices using the kernel driver:
==================================
0000:14:00.2 'Ethernet Controller for 1GbE Other network devices:
0000:14:00.3 'Ethernet Controller for 1GbE
=====================
<none>
```

6.6 DPDK Forwarding Application

Downloaded DPDK comes with few sample applications in the `dpdk/examples` directory. We used L2 forwarding application for our test methodology.

1. To build the L2 Forward application execute:

```
# cd ${RTE_SDK}/examples/l2fwd-jobstats
# export RTE_TARGET=x86_64-native-linuxapp-gcc
# make
```

2. To run the application:

```
./build/l2fwd -c 0x6 -n 2 -- -p 0x3
```

More details on its usage available at http://dpdk.org/doc/guides/sample_app_ug/l2_forward_real_virtual.html.

7.0 OvS and OvS with DPDK Setup

This section assumes that Fedora 22 is already installed using the steps detailed in Intel® ONP Reference Architecture Guide. Then follow the steps in Section 6 (until Section 6.5) to configure the host, install DPDK and bind the 2 ports with igb_uio driver.

7.1 Install OvS

1. Clone the OvS repository from github and go to the OvS directory to run installation

```
# git clone https://github.com/openvswitch/ovs.git
# git reset --hard
# ./boot.sh
# ./configure --with-dpdk=/root/dpdk/x86_64-native-linuxapp-gcc \
              CFLAGS=”-Ofast -g”
# make 'CFLAGS=-g -Ofast -march=native' -j8
```

7.2 Remove and Terminate Previous Run of 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
```

7.3 Initialize new OvS Database

1. Initialize the new OVS database:

```
# export OVS_DIR=/root/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
```

7.4 Start and Tune 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
```

2. 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
```

7.5 Create the Ports

1. Create a 2-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 show
```

7.6 Add the Port Flows

1. Clear current flows:

```
# 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
```

8.0 vE-CPE Setup with Intel® ONP

8.1 Intel® ONP Reference Architecture

Intel® Open Network Platform (Intel® ONP) is a Reference Architecture that provides engineering guidance and ecosystem-enablement support to encourage widespread adoption of SDN and NF) solutions in Telco, Enterprise, and Cloud applications. With OpenStack as the orchestrator and Open vSwitch combined with DPDK as the data plane accelerator, it provides the software stack necessary to provision and deploy user specific VNFs to build the required use case.

This report focuses on using Intel® ONP reference architecture to provision and deploy the vE-CPE use case. The figure below shows the setup used for vE-CPE benchmarking. The reference architecture setup and configuration information is available as the Intel® ONP 2.0 Reference Architecture Guide at 01.org.

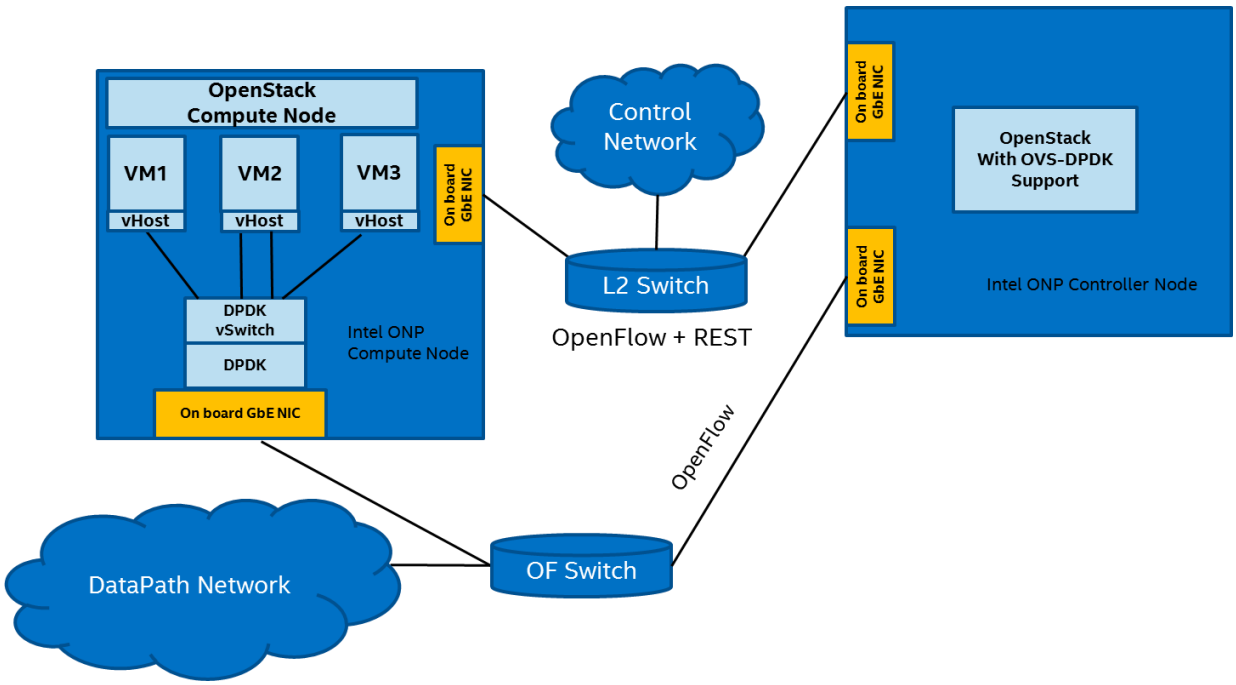

Figure 8-1 Intel® ONP test setup for vE-CPE use case

8.2 Provision Intel® ONP Setup for vE-CPE Use Case

In Figure 8-1 the Compute Node is the DUT. The Intel® ONP setup has to be customized with the configuration changes listed below in order to implement a vE-CPE use case that can be benchmarked using Ixia traffic generator.

8.2.1 Two Tenant Ports with VLAN Overlay

Intel® ONP provisions the platform exposing the host's tenant interface, management interface and Internet interface to OpenStack Neutron agent. This benchmarking methodology with Ixia traffic generator requires two

tenant network interfaces to measure packets sent and received across two ports. The following changes are required to enable two tenant ports with VLAN as overlay network:

1. Follow the instructions from Intel® ONP Reference Architecture Guide to configure `onps_config file`, execute the script `prepare_system.sh`, reboot and update using "`dnf update -y`".
2. Before executing the `prepare_stack.sh` script, update the OpenStack `local.conf` file with following under OvS with DPDK related configuration options:

```
OVS_BRIDGE_MAPPINGS=physnet1:br-<interface1>,physnet2:br-<interface2>
ML2_VLAN_RANGES=physnet1:<VLANRange1Begin>:<VLANRange1End>,physnet2:
<VLANRange2Begin>:<VLANRange2End>
```

   As an example, assume that:

   - eth0 and eth1 are the two tenant interfaces that are used for connecting to traffic generator
   - The VLAN ranges eth0 interface supports is 1000 to 1010, VLAN ranges supported by eth1 interface is 2000 to 2010
   - The updates to `local.conf` file would be like below:

```
OVS_BRIDGE_MAPPINGS=physnet1:br-eth0,physnet2:br-eth1
ML2_VLAN_RANGES=physnet1:1000:1010,physnet2:2000:2010
```

   Note: These changes are case sensitive, wrong updates or typos would result in failure of installing OpenStack services.

The networking-ovs-dpdk ML2 plugin supports configuring two tenant ports as a special functionality added for deployments with accelerated vSwitch, OvS with DPDK, configuration. The vanilla DevStack with Open vSwitch agent does not support deploying OpenStack with multiple tenant ports.

More details on OVS_BRIDGE_MAPPINGS parameter can be found in the networking-ovs-dpdk site:
https://github.com/openstack/networking-ovs-dpdk/blob/master/doc/source/usage.rst

Following list shows configuration of four ports of the system:

- Port 0: management network, assume low data rate, e.g. 10/100/1000 Mbps and not used for data-plane testing
- Port 1: Internet traffic, mainly used for provisioning and deploying OpenStack
- Port 2: Customer Premises LAN (1GbE), called Tenant port 1with VLAN tag of 1001
- Port 3: To Service Provider WAN (1GigE), called Tenant port 2 with VLAN tag of 2001

Traffic flow:

- Assume all traffic goes through all virtual appliances.
- Traffic is offered on (only) 1 GbE physical interface.

Note: This option to have two tenant ports is only available by using networking-ovs-dpdk OpenStack ML2 plugin. Only one tenant port can be configured with VLAN overlay using Neutron and open vSwitch as the Neutron agent.

8.2.2 Enhanced Platform Awareness Features of OpenStack

There are many features provided by OpenStack to take advantage of platform features using EPA. Intel® ONP 2.0 exposes following Enhanced Platform Awareness features to OpenStack and Open vSwitch that were used in this report to enhance performance:

Dedicated Physical CPUs

Depending on the workload being executed, the end user may wish to have control over how the guest VM uses hyper-threads. To maximize cache efficiency, the guest may wish to be pinned to thread siblings. Conversely, the guest may wish to avoid thread siblings (i.e. only pin to 1 sibling) or even avoid hosts with threads entirely. This level of control is of particular importance to NFV deployments which care about maximizing cache efficiency of vCPUs.

OpenStack flavor provides extra specs below to provide this support.

```
$ nova flavor-key <flavor id> set "hw:cpu_policy=dedicated"
```

If the policy is set to `dedicated` then the guest virtual CPUs will be strictly pinned to a set of host physical CPUs.

Scheduler Filters

The Filter Scheduler supports filtering and weighting to make informed decisions on where a new instance should be created. Filter Scheduler iterates over all found Compute Nodes, evaluating each against a set of filters. The list of resulting hosts is ordered by weighers. The Scheduler then chooses hosts for the requested number of instances, choosing the most weighted hosts. The following filters are used in Intel® ONP scripts:

- **RamFilter** filters hosts by their RAM. Only hosts with sufficient RAM to host the instance are passed.
- **ComputeFilter** passes all hosts that are operational and enabled
- **AvailabilityZoneFilter** passes hosts matching the availability zone specified in the instance properties. Use a comma to specify multiple zones. The filter will then ensure it matches any zone specified.
- **ComputeCapabilitiesFilter** checks that the capabilities provided by the host compute service satisfy any extra specifications associated with the instance type. It passes hosts that can create the specified instance type.
- **ImagePropertiesFilter** filters hosts based on properties defined on the instance's image. It passes hosts that can support the properties specified on the image used by the instance.
- **NUMATopologyFilter** filters hosts based on the NUMA topology requested by the instance, if any.
- **PciPassthroughFilter** schedules instances on a host if the host has devices to meet the device requests in the 'extra_specs' for the flavor.

More details on scheduler filters can be found at
http://docs.openstack.org/developer/nova/filter_scheduler.html.

OVS Core Pinning

Networking-OVS-DPDK ML2 plugin provides options to configure and set affinity to OVS processes. The Intel® ONP scripts by default set the below options to better pin OvS processes to physical CPUs:

1. OVS_CORE_MASK: CPU cores are selected according to this hex value to pin OvS processes. The default value used in scripts is 0x02.
2. OVS_PMD_CORE_MASK: The mask in hex format for the Poll Mode Drivers (PMD) threads of OvS set in its database. The default value used in the scripts is 0x04.

More details about these options are detailed in Networking-OVS-DPDK ML2 Plugin documentation available at https://github.com/openstack/networking-ovs-dpdk/blob/master/doc/source/usage.rst.

8.2.3 VNF Emulation with DPPD

Intel® Data Plane Performance Demonstrators (DPPD) or currently called PROX (Packet pROcessing eXecution engine) tool is a DPDK based Linux user space application that can be used to emulate variety of virtual network functions. It consists of many example applications intended to:

- develop best practices for implementing high-performance packet processing solutions on Intel® platforms
- develop TCO models for Telco customers evaluating Intel-based networking solutions
- investigate the impact of new Intel® compute, network and storage technologies on performance
- characterize performance of VNFs in the context of key industry NFV and SDN use cases
- develop optimal system architectures and configurations for multi-VM packet processing environments

This report showcases use of DPPD as L2 forwarding application that modifies the packet header with the destination MAC address and forwards the packets to next hop. The tools shows performance statistics of the VNF indicating:

- Core utilization
- Transmitted and Received packets
- Packet drop inside the VNF

More command line options for the tool are available at: https://01.org/intel-data-plane-performance-demonstrators/documentation/prox-documentation

DPPD version 017 is used for which the corresponding supported version of DPDK 2.0 is used for this effort. However the default package configurations does not suffice this use case. A L3 forwarding application is provided with the default package, the user can configure the L2 forwarding application by following the steps detailed below.

8.2.4 Configure VNF for DPPD

1. Download the DPDK-2.0.0 software from dpdk.org
```
# wget http://www.dpdk.org/browse/dpdk/snapshot/dpdk-2.0.0.tar.gz
```
2. Go to the dpdk directory and run the following:
```
# make install T=x86_64-native-linuxapp-gcc
# cd x86_64-native-linuxapp-gcc
```
3. Edit the config file (vim .config) and set the configuration options:
```
CONFIG_RTE_BUILD_COMBINE_LIBS=n
CONFIG_RTE_LIBRTE_VHOST=n
CONFIG_RTE_LIBRTE_VHOST_USER=y
```
4. Save the config file and run make:
```
# EXTRA_CFLAGS=”-g -Ofast”
# make -j2
```
5. Mount the 1GB HugePage and 2MB HugePage:
```
# mkdir -p /mnt/huge_2mb
# mount -t hugetlbfs nodev /mnt/huge_2mb -o pagesize=2MB
```
6. Check that HugePages are mounted:
```
# mount nodev on /mnt/huge_2mb type hugetlbfs (rw,relatime,pagesize=2MB)
```
7. Allocate 1024 2MB hugepages
```
# echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr-hugepages
# cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr-hugepages
```
8. Remove the following Linux modules and load the modules for OVS:
```
# rmmod ixgbe
# rmmod igb_uio
# rmmod cuse
# rmmod fuse
# rmmod uio
# rmmod eventfd_link
# rmmod ioeventfd
# modprobe uio
# insmod $DPDK_BUILD/kmod/igb_uio.ko
```
9. Check the PCI ID for the 1GbE NIC ports:
```
# lspci | grep Ethernet
```

```
0000:00:03.0 Ethernet controller: Intel Corporation Ethernet Controller for
1GbE
0000:00:04.0 Ethernet controller: Intel Corporation Ethernet Controller for
1GbE
```

10. Bind the ports:

```
# python $DPDK_DIR/tools/dpdk_nic_bind.py --bind=igb_uio 0000:00:03.0
# python $DPDK_DIR/tools/dpdk_nic_bind.py --bind=igb_uio 0000:00:04.0
# python $DPDK_DIR/tools/dpdk_nic_bind.py -status

Network devices using the DPDK-compatible driver:
============================================
0000:00:03.0 'Ethernet Controller for 1GbE +' drv=igb_uio unused=
0000:00:04.0 'Ethernet Controller for 1GbE +' drv=igb_uio unused=
Network devices using the kernel driver:
==================================
<none>
```

8.2.5 Configure and Run DPPD as Unidirectional L2 Forwarding Application

1. Download and unzip the DPPD-v017 from 01.org: https://01.org/intel-data-plane-performance-demonstrators/downloads/bng-application-v017
2. Create a DPPD configuration file for L2 forwarding from existing L3 forwarding configuration file:

```
# cd dppd-bng-v017/config
# cp handle_none.cfg handle_l2fwd.cfg
```

3. Edit the contents of `handle_l2fwd.cfg` to look as following:

```
[eal options]
-n=4 ; force number of memory channels
no-output=no ; disable DPDK debug output

[port 0]
name=if0
mac=hardware
promiscuous=no
[port 1]
name=if1
mac=hardware
promiscuous=no

[defaults]
mempool size=4K

[global]
start time=5
name=Forwarding (2x)

[core 0]
mode=master
```

```
[core 1]
name=forward
task=0
mode=l2fwd
rx port=if0
tx port=if1
dst mac=
```

4. Update the next hop MAC address in the field "`dst mac=`". Note, for task=0, update the destination MAC address for packets received at interface `if0` and transmitted out of interface `if1`.
5. Start DPPD L2 forward application

```
# cd dppd-BNG-0v17/
# ./build/dppd -f config/handle_l2fwd.cfg
```

6. On a successful startup, the screen would look like:

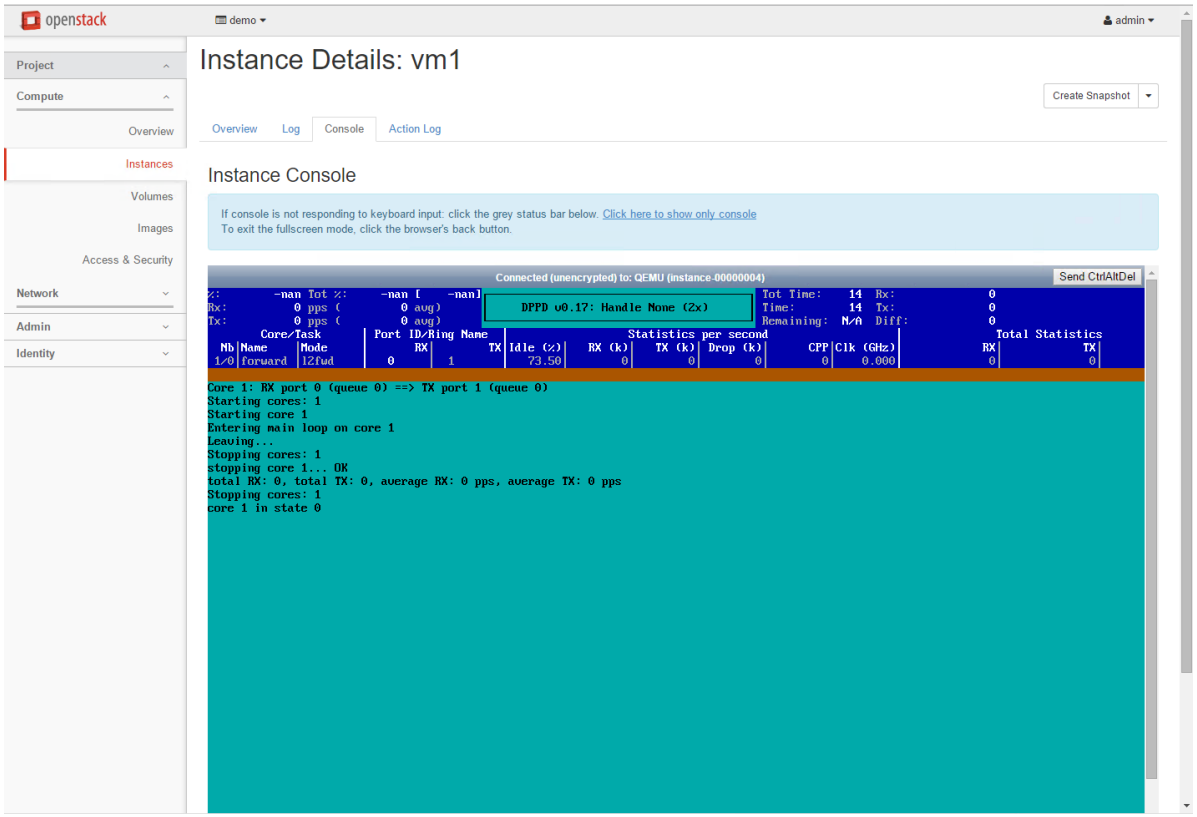

Figure 8-2 DPPD-v017 startup screen for unidirectional configuration

8.2.6 Configure and Run DPPD as Bidirectional L2 Forwarding Application

DPPD requires at least one virtual core for itself to take care of its associated threads for video processing, packet counters etc. Given in our methodology the VNF only has only 2 virtual cores at maximum, only one virtual core is available for packet forwarding operations. DPPD provides concept of "tasks" in order for multiple operations to be done on a same virtual core. Hence, we divide each flow of the East-West traffic across the VMs to be assigned a separate tasks. The example configuration below details how to configure DPPD L2 FWD application for bidirectional traffic:

1. Download and unzip the DPPD-v17 from 01.org: https://01.org/intel-data-plane-performance-demonstrators/downloads/bng-application-v017
2. Create a DPPD configuration file for L2 forwarding from existing L3 forwarding configuration file:

```
# cd dppd-bng-v017/config
# cp handle_none.cfg handle_l2fwd.cfg
```

3. Edit the contents of `handle_l2fwd.cfg` to look as following:

```
[eal options]
-n=4 ; force number of memory channels
no-output=no ; disable DPDK debug output

[port 0]
name=if0
mac=hardware
promiscuous=no
[port 1]
name=if1
mac=hardware
promiscuous=no

[defaults]
mempool size=4K

[global]
start time=5
name=Forwarding (2x)

[core 0]
mode=master

[core 1]
name=forward
task=0
mode=l2fwd
rx port=if0
tx port=if1
dst mac=

[core 1]
name=forward
task=1
mode=l2fwd
rx port=if1
tx port=if0
dst mac=
```

4. Update the next hop MAC address in the field "`dst mac=`" with following considerations:
   - For task=0, update the destination MAC address for packets received at interface `if0` and transmitted out of interface `if1`
   - For task=1, update the destination mac address for packets received at interface `if1` and transmitted out of interface `if0`

Note: The "How To" of configuring the "dst_mac=" field is detailed in corresponding sections below for both PHY-OVS-VM-OVS-PHY case and vE-CPE use case.

5. Start DPPD L2 forward application

```
# cd dppd-BNG-0v17/
# ./build/dppd -f config/handle_l2fwd.cfg
```

6. On a successful startup, the screen would look like:

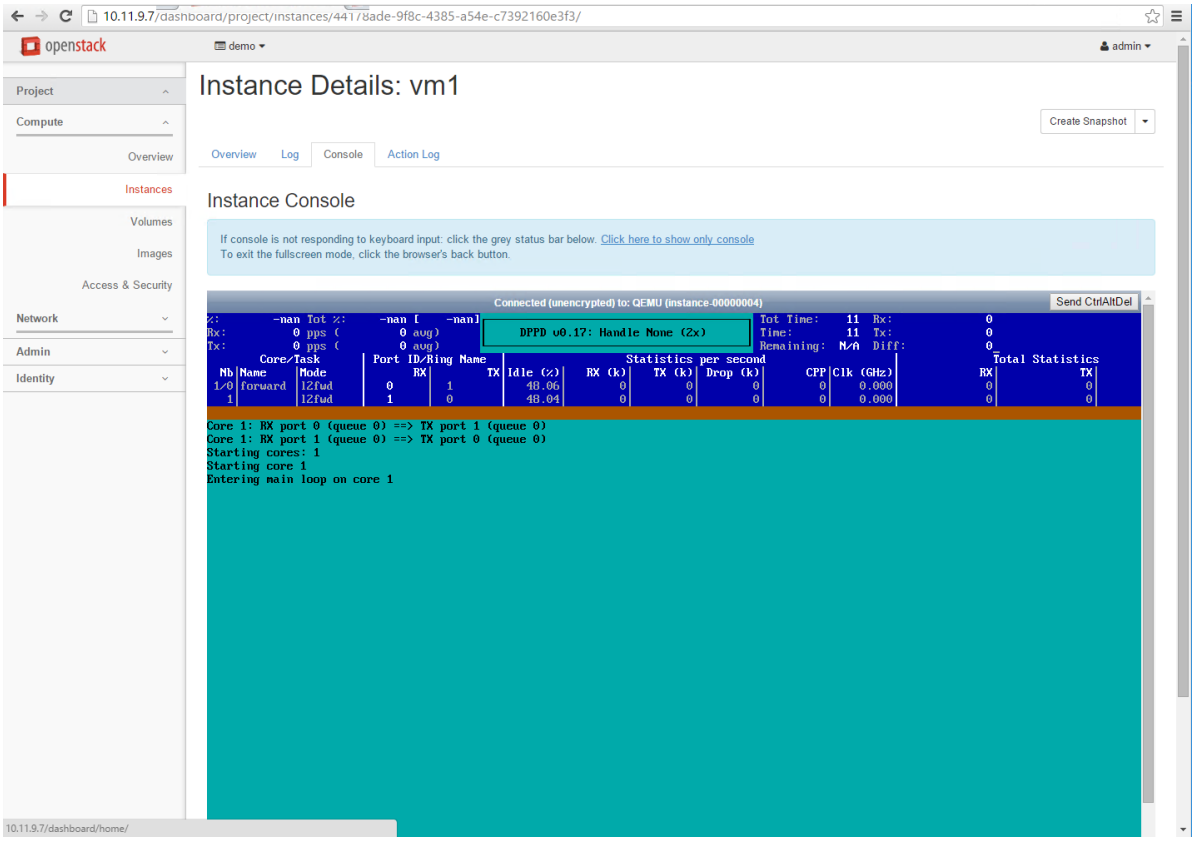

Figure 8-3 DPPD-v017 startup screen for bidirectional configuration

Note: There are two l2fwd tasks displayed with DPPD since there this is configured for a bidirectional flow.

8.3 PHY-OVS-VM-OVS-PHY Setup with Intel® ONP

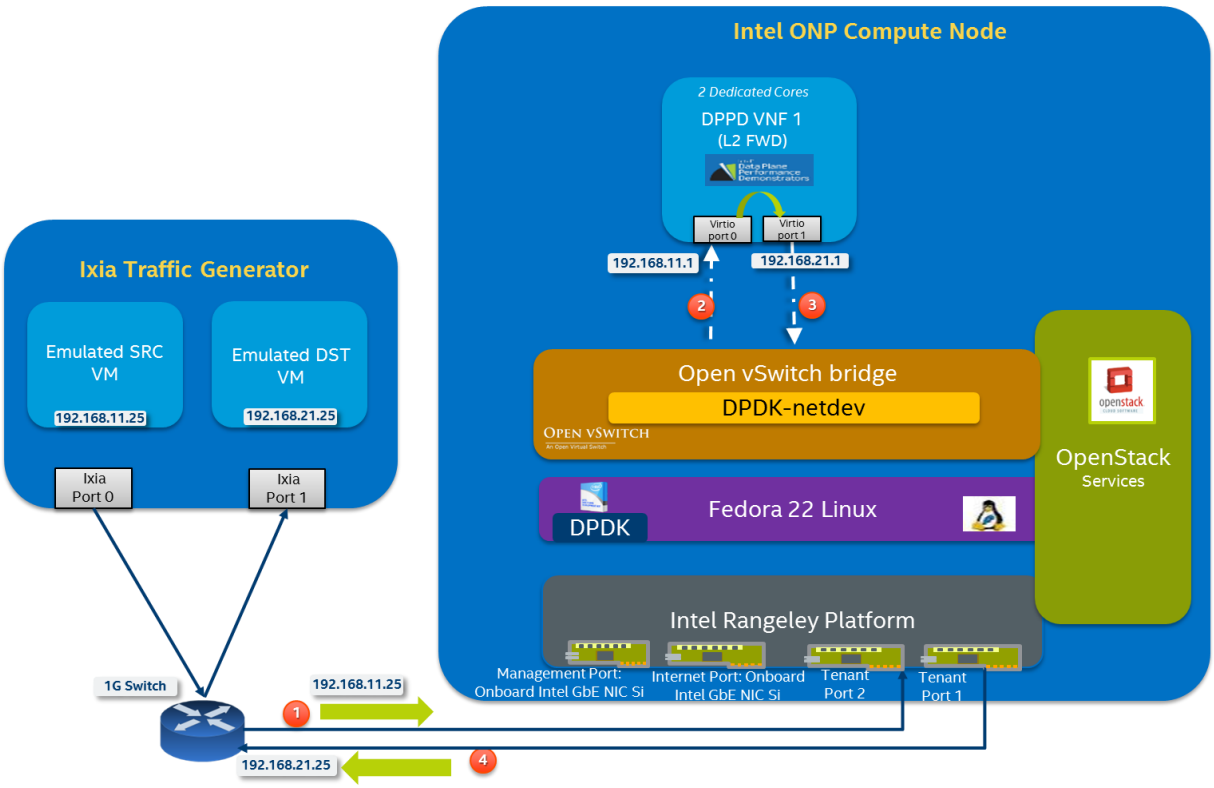

Figure 8-4 One VM Traffic flow

Note: This section assumes that user has successfully provisioned the platform with Intel® ONP scripts and followed the updates in 8.2.1 Two Tenant Ports with VLAN Overlay.

The following steps describe how to create a DPPD based VNF using Fedora 20 VM image with custom options like flavor, and aggregate/availability zone using OpenStack commands.

1. Log in as **stack** user.

   Note: Some OpenStack commands (e.g., Keystone and aggregate-related) can only be used by the admin user, while others (e.g., Glance, Nova, and those that are Neutron-related) can be used by other users, but with limited visibility.

   Note: DevStack provides a built-in tool, openrc, located at /home/stack/devstack/ to source an OpenStack user credential to the shell environment.

2. Source the admin credential:

   ```
   $ source /home/stack/DevStack/openrc admin demo
   ```

   Note: OpenStack commands will thereafter use the admin credential.

3. Create an OpenStack Glance image. Glance is the OpenStack component that manages VM images. A VM image file should be ready in a location accessible by OpenStack. The following command creates an OpenStack image from an existing image file. The image is used as a template for creating new VMs:

   ```
   $ glance image-create --name <image-name-to-create> --visibility=public --
   container-format=bare --disk-format=<format> --file=<image-file-path-name>
   ```

   The following example shows the image file, fedora20-x86_64-basic.qcow2, the command creates a Glance image named fedora-basic with a qcow2 format for the public that any tenant can use:

```
$ glance image-create --name fedora-basic --visibility=public –container-
format=bare --disk-format=qcow2 --file=/mnt/nfs/openstack/images/fedora20-
x86_64-basic.qcow2
```

4. Create the host aggregate and availability zone. First find out the available hypervisors, and then use this information to create an aggregate/ availability zone:

```
$ nova hypervisor-list
$ nova aggregate-create <aggregate-name> <zone-name>
$ nova aggregate-add-host <aggregate-name> <hypervisor-name>
```

   The following example creates an aggregate named aggr-g06 with one availability zone named `zone-g06` and the aggregate contains one hypervisor named sdnlab-g06:

```
$ nova aggregate-create aggr-g06 zone-g06
$ nova aggregate-add-host aggr-g06 sdnlab-g06
```

5. Create a flavor. Flavor is a virtual hardware configuration for the VMs; it defines the number of virtual CPUs, size of the virtual memory, disk space, etc.

   The following command creates a flavor named onps-flavor with an ID of 1001, 1024 Mb virtual memory, 4 Gb virtual disk space, and 1 virtual CPU:

```
$ nova flavor-create onps-flavor 1001 1024 4 1
```

6. With the OvS-DPDK compute node with a vhost-user, a large memory page size should be configured for the OpenStack flavor for creating instances. This can be done in two steps: first create a flavor, and then modify it to allow a large memory page size.

   The following commands create a flavor named largepage-flavor with an ID of 1002, 2048 Mb virtual memory, 20 Gb virtual disk space, 2 virtual CPUs, and large memory page size:

```
$ nova flavor-create largepage-flavor 1002 2048 20 2
$ nova flavor-key 1002 set "hw:mem_page_size=large"
```

   This flavor configuration was used to create instances hosted by OvS-DPDK compute node with a vhost-user.

7. We configure the flavor to allow virtual CPUs in the VM to have a dedicated physical core so as to facilitate a utilization of a complete physical core for every virtual core in the VM. The following command configures the flavor to have dedicated CPU policy.

```
$ nova flavor-key 1002 set "hw:cpu_policy=dedicated"
```

8. Source the demo user credential. Note that OpenStack commands will continue to use this demo credential:

```
$ source /home/stack/DevStack/openrc demo demo
```

9. The default Intel® ONP scripts configure one physical network on Tenant network with one physical port. Our use case requires the host to have two physical networks on tenant network to be configured for tenant network. OpenStack allows us to do this by configuring neutron networks to be created with additional parameters.
10. Following the example from 8.2.1 Two Tenant Ports with VLAN Overlay., tenant port eth0 would have VLAN ranges 1000 to 1010 and tenant port eth1 would have VLAN ranges 2000 to 2010. For the one VM case, we create two networks with following settings:
    - net-phys1-1001 on physical network physnet1 that allows traffic with VLAN tag of 1001
    - net-phys2-2001 on physical network physnet2 that allows traffic with VLAN tag of 2001
11. The following commands help choose the physical network on a specific VLAN for a tenant network:

```
$ neutron net-create <network name> --provider:network_type vlan --
provider:physical_network <physical network name> --provider:segmentation_id
<VLAN ID>
$ neutron subnet-create --name <subnet_name> <network-name> <net-ip-range>
```

12. To configure with the required mapping:

```
$ neutron net-create net-phys1-1001 --provider:network_type vlan -- \
provider:physical_network physnet1 --provider:segmentation_id 1001
```

```
$ neutron subnet-create --name subnet-phys1-1001 net-phys1-1001 \
192.168.11.0/24
$ neutron net-create net-phys2-2001 --provider:network_type vlan -- \
provider:physical_network physnet2 --provider:segmentation_id 2001
$ neutron subnet-create --name subnet-phys2-2001 net-phys1-1002 \
192.168.21.0/24
```

13. Create an instance (VM) for the tenant demo as follows:

    Get the name and/or ID of the image, flavor, and availability zone to be used for creating the instance:

```
$ glance image-list
$ nova flavor-list
$ nova availability-zone-list
$ neutron net-list
```

    Launch the instance (VM) using information from the previous step:

```
$ nova boot --image <image-id or image-name> --flavor <flavor-id or flavor-
name> --availability-zone <zone-name> --nic net-id=<network-id> <instance-
name>
```

14. Deploy instance (VM) on security disabled ports

```
$ nova boot --flavor=<flavor-id or flavor-name> --image=<image-id or image-
name> --nic port-id=$(neutron port-show -f value -F id <port_name-1>) --nic
port-id=$(neutron port-show -f value -F id <port_name-2>) <instance-name>
```

    The new VM should be up and running in a few minutes.

15. Log in to the OpenStack dashboard using the demo user credential; click **Instances** under "Project" in the left pane; the new VM should show in the right pane. Click the instance name to open the **Instance Details** view, and then click **Console** in the top menu to access the VM as show below.

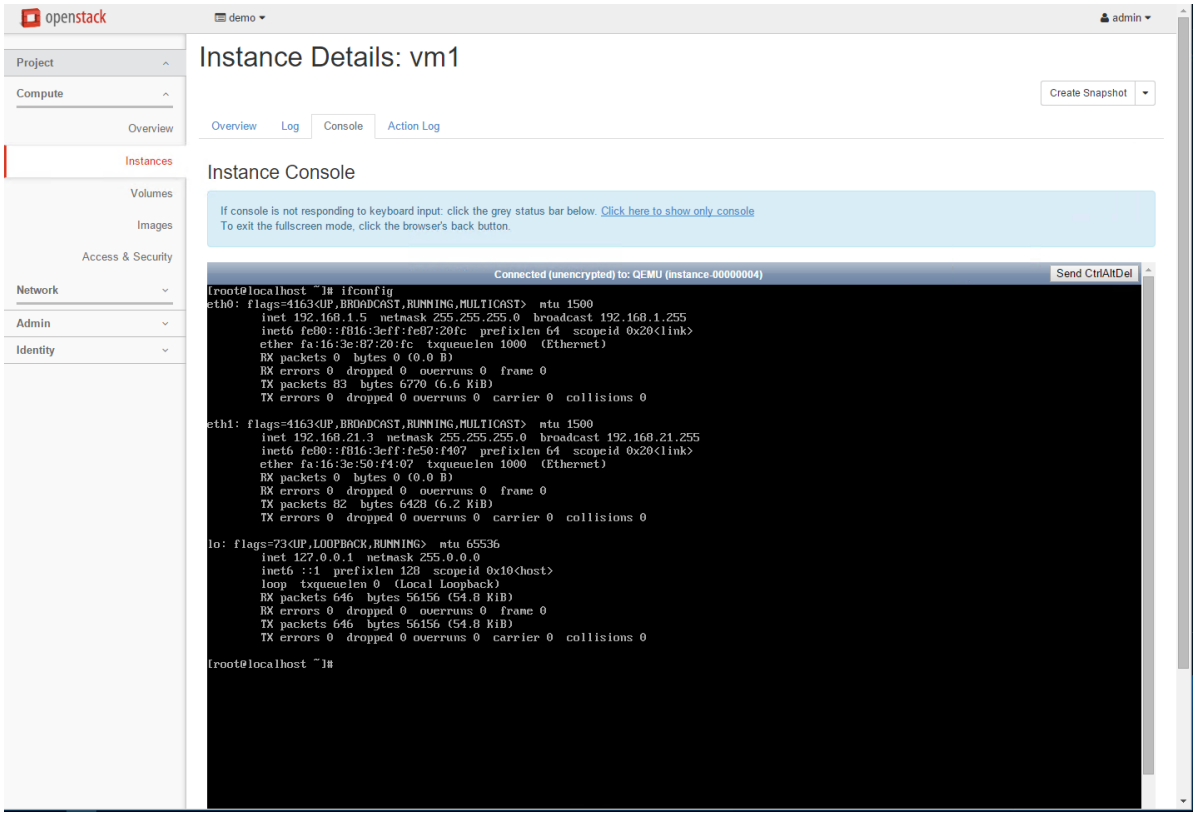

Figure 8-5 OpenStack Instance Console

16. Follow the steps detailed in 8.2.4, 8.2.5, 8.2.6 to provision the VM with DPPD tool with following considerations in the `handle_l2fwd.cfg` file:
    - For unidirectional flow the "dst mac=" value should be 192.168.21.25, MAC address of the emulated destination VM of Ixia, from Figure 8-4
    - For bidirectional flow, from Figure 8-4:
        - Traffic from 192.168.11.25 would hit port `eth0`, so the "`dst mac`=" value of "`task=0`" should be 192.168.21.25
        - Traffic from 192.168.12.25 would hit port eth0, so the "`dst mac`=" value of "`task=1`" should be 192.168.11.25

    Note: For benchmarking purpose, ensure to follow the steps listed in 9.2 Educating OvS of TCP/IP-less DPDK Based VNFs before starting DPPD application.

17. Execute the application (from 8.2.5 or 8.2.6) as a L2 forwarding VNF

With the correct setup, as the traffic from packet generator reaches the virtio ports of the VM, the packet counters in DPPD application will be live indicate Rx, Tx and packet drop statistics.

8.4 vE-CPE Setup with Intel® ONP

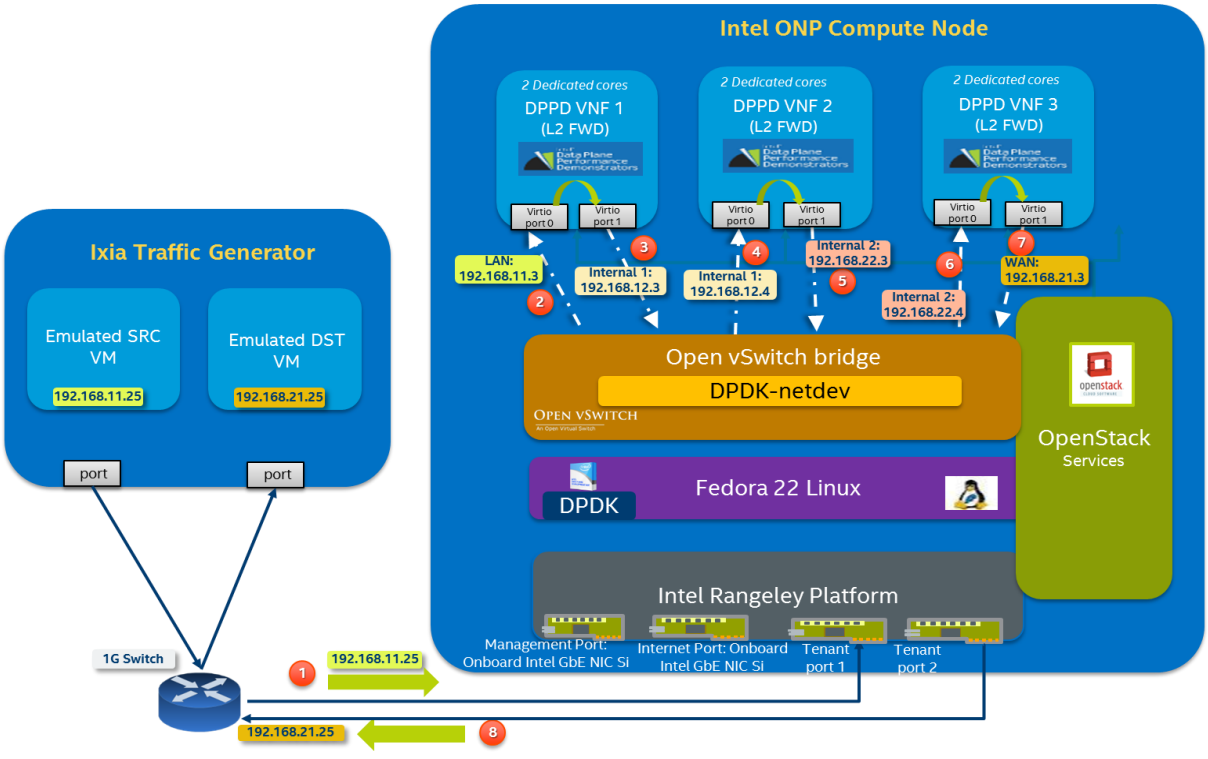

Figure 8-6 Unidirectional traffic flow between Ixia and vE-CPE setup

Note: This section assumes that user has successfully provisioned the platform with Intel® ONP scripts and followed the updates in 8.2.1 Two Tenant Ports with VLAN Overlay.

The vE-CPE use case is provisioned with Intel® ONP and the VNFs are deployed using OpenStack. A traditional vE-CPE equipment is ideally placed between the ISP infrastructure connected with a WAN and customer devices

connected with a LAN. The vE-CPE setup using Intel® ONP showcases similar network setup using OpenStack Neutron to have separate networks for LAN, WAN and set of internal network for the VNFs to communicate with each other. We recreate a vE-CPE functionality by using DPPD L2 forwarding application to emulate applications like deep packet inspection functionality as an example. Figure 8-6 showcases the traffic flow between these different networks. The "Tenant Port 1" can be assumed to be connected to a LAN and the "Tenant Port 2" can be assume to be connected to a WAN.

To have correct interoperability between VNFs, the traffic from packet generator to traverse across VNFs and back to packet generator and to showcase a working vE-CPE use case, it's imperative to configure the Neutron networks and provision the VNFs with DPPD using the steps detailed in following sections.

8.4.1 VNF Considerations for Intel® Atom™ processor C2758 Platform

The Intel® Atom™ Processor C2758 Platform has 8 cores available with which the user has to configure their VNFs to get the best performance. For the vE-CPE use case in this benchmarking report the 8 cores on the host DUT were dedicated in following way:

Table 8-1 Intel® Atom™ cores assignment in the vE-CPE setup

Application/Process	Number of Cores
OvS PMD	1
OvS vSwitchd	1
VNF1 with DPPD L2 forward application	2
VNF2 with DPPD L2 forward application	2
VNF3 with DPPD L2 forward application	2

Using the "extra specs" property of flavors, OpenStack provides a dedicated CPU policy using which user can request a specified set of dedicated cores for the VNF. The configuration details are described in 8.4.2 Generic OpenStack Configuration. However this does not guarantee that the cores are exclusively dedicated for the VNF.

One way to overcome this limitation is to enable core isolation in the boot options. Unfortunately, core isolation could not be used for this release since OpenStack Liberty deployment fails with core isolation enabled.

Each VNF used were provisioned and deployed with following system configurations:

Table 8-2 System configurations for VNFs in vE-CPE setup

Number of Virtual Cores	2
Memory (RAM)	2 GB
Virtual hard disk	20 GB
Operating System	Fedora 20
DPPD	version 17, available from https://01.org/sites/default/files/downloads/intel-data-plane-performance-demonstrators/dppd-bng-v017.zip
Hugepage size	2 MB
Number of Hugepages	1024

8.4.2 Generic OpenStack Configuration

The following steps describe how to create a DPPD based VNF using Fedora 20 VM image with custom options like flavor, and aggregate/availability zone using OpenStack commands.

1. Log in as **stack** user.

   Note: Some OpenStack commands (e.g., Keystone and aggregate-related) can only be used by the admin user, while others (e.g., Glance, Nova, and those that are Neutron-related) can be used by other users, but with limited visibility.

   Note: DevStack provides a built-in tool, openrc, located at /home/stack/devstack/ to source an OpenStack user credential to the shell environment.

2. Source the admin credential:

   ```
   $ source /home/stack/DevStack/openrc admin demo
   ```

   Note: OpenStack commands will thereafter use the admin credential.

3. Create an OpenStack Glance image. Glance is the OpenStack component that manages VM images. A VM image file should be ready in a location accessible by OpenStack. The following command creates an OpenStack image from an existing image file. The image is used as a template for creating new VMs:

   ```
   $ glance image-create --name <image-name-to-create> --visibility=public --
   container-format=bare --disk-format=<format> --file=<image-file-path-name>
   ```

   The following example shows the image file, fedora20-x86_64-basic.qcow2, the command creates a Glance image named fedora-basic with a qcow2 format for the public that any tenant can use:

   ```
   $ glance image-create --name fedora-basic --visibility=public –container-
   format=bare --disk-format=qcow2 --file=/mnt/nfs/openstack/images/fedora20-
   x86_64-basic.qcow2
   ```

4. Create the host aggregate and availability zone. First find out the available hypervisors, and then use this information to create an aggregate/ availability zone:

   ```
   $ nova hypervisor-list
   $ nova aggregate-create <aggregate-name> <zone-name>
   $ nova aggregate-add-host <aggregate-name> <hypervisor-name>
   ```

   The following example creates an aggregate named aggr-g06 with one availability zone named zone-g06 and the aggregate contains one hypervisor named sdnlab-g06:

   ```
   $ nova aggregate-create aggr-g06 zone-g06
   $ nova aggregate-add-host aggr-g06 sdnlab-g06
   ```

5. Create a flavor. Flavor is a virtual hardware configuration for the VMs; it defines the number of virtual CPUs, size of the virtual memory, disk space, etc.

   The following command creates a flavor named onps-flavor with an ID of 1001, 1024 Mb virtual memory, 4 Gb virtual disk space, and 1 virtual CPU:

   ```
   $ nova flavor-create onps-flavor 1001 1024 4 1
   ```

6. With the OvS-DPDK compute node with a vhost-user, a large memory page size should be configured for the OpenStack flavor for creating instances. This can be done in two steps: first create a flavor, and then modify it to allow a large memory page size.

   The following commands create a flavor named largepage-flavor with an ID of 1002, 2048 Mb virtual memory, 20 Gb virtual disk space, 2 virtual CPUs, and large memory page size:

   ```
   $ nova flavor-create largepage-flavor 1002 2048 20 2
   $ nova flavor-key 1002 set "hw:mem_page_size=large"
   ```

   This flavor configuration was used to create instances hosted by OvS-DPDK compute node with a vhost-user.

7. We configure the flavor to allow virtual CPUs in the VM to have a dedicated physical core so as to facilitate a utilization of a complete physical core for every virtual core in the VM.

   The following command configures the flavor to have dedicated CPU policy.

```
$ nova flavor-key 1002 set "hw:cpu_policy=dedicated"
```

8.4.3 VNF Configuration in vE-CPE

In a physical vE-CPE box, every VNF has each of its interfaces on a different physical network with its own VLAN tag. In order to showcase the vE-CPE with varying physical networks the following topology is used in our deployment:

Table 8-3 VNF Network Configuration

	VNF1 eth0	VNF1 eth1	VNF2 eth0	VNF2 eth1	VNF3 eth0	VNF3 eth1
Physical Network	physnet1	physnet1	physnet1	physnet2	physnet2	physnet2
Physical Network Name	net-phys1-1001	net-phys1-1002	net-phys1-1002	net-phys2-2001	net-phys2-2001	net-phys2-2002
VLAN Tag	1001	1002	1002	2001	2001	2002
MAC address	00:00:00:00:00:10	00:00:00:00:00:11	00:00:00:00:00:20	00:00:00:00:00:21	00:00:00:00:00:30	00:00:00:00:00:31
IP Address	192.168.11.3	192.168.12.3	192.168.12.4	192.168.22.3	192.168.22.4	192.168.21.3

Note that the MAC addresses and IP addresses listed here are only for reference, with OpenStack deployment Neutron takes care of both MAC address and IP address assignments. The IP addresses are assigned with in the subnets configured for corresponding physical networks.

After completing the steps under 8.4.2 Generic OpenStack Configuration follow the steps detailed here for the 3 VNF setup:

1. Source the demo user credential. Note that OpenStack commands will continue to use this demo credential:

```
$ source /home/stack/DevStack/openrc admin demo
```

2. The default Intel® ONP scripts configure one physical network on Tenant network with one physical port. Our use case requires the host to have two physical networks on tenant network to be configured for tenant network. OpenStack allows us to do this by configuring neutron networks to be created with additional parameters.

   The following commands help choose the physical network on a specific VLAN for a tenant network:

```
$ neutron net-create <network name> --provider:network_type vlan --
provider:physical_network <physical network name> --provider:segmentation_id
<VLAN ID>
$ neutron subnet-create --name <subnet_name> <network-name> <net-ip-range>
```

   Example to configure with the required mapping:

```
$ neutron net-create net-phys1-1001 --provider:network_type vlan -- \
provider:physical_network physnet1 --provider:segmentation_id 1001
$ neutron subnet-create --name subnet-phys1-1001 net-phys1-1001 \
192.168.11.0/24
$ neutron net-create net-phys2-2001 --provider:network_type vlan -- \
provider:physical_network physnet2 --provider:segmentation_id 2001
$ neutron subnet-create --name subnet-phys2-2001 net-phys1-1002 \
192.168.21.0/24
```

3. Follow the network setup from Table 8-3 to configure required physical networks with their corresponding VLAN tags and subnet IP addresses.
4. Create an instance (VM) for the tenant demo using Horizon UI by logging in as **demo** user.

   Note: If you need more details on using Horizon UI, check the Appendix B section of Intel® ONP Reference Architecture Guide.

5. Click the **Instances** tab under **Project > Compute** in the left pane. Click the **Launch Instance** tab at the upper-right corner in the new window, and then select the desired availability zone, instance name, flavor, and instance boot source from the respective drop-down boxes.

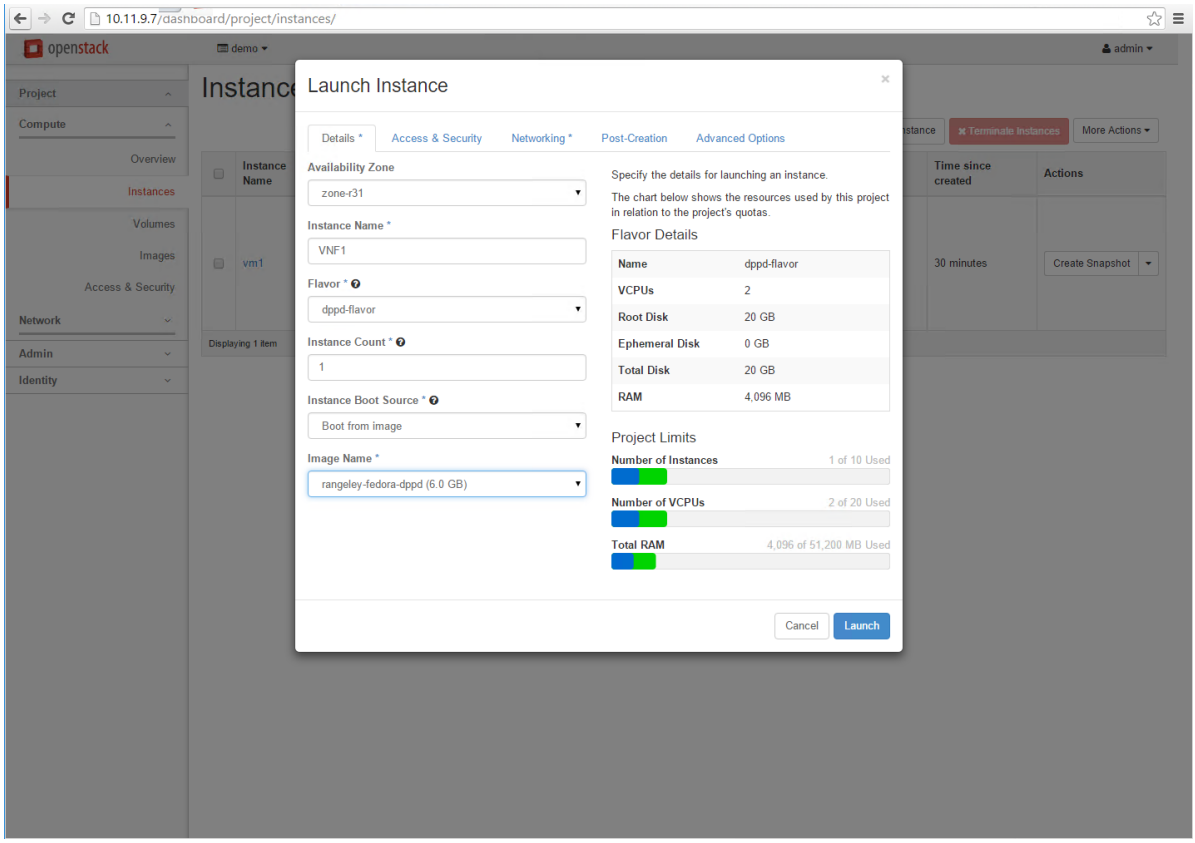

Figure 8-7 OpenStack Instances UI – Launching Instance

6. Click the **Networking** tab, and then select one or more networks for the instance. All networks, including the default network, private, and newly created ones, are available for selection. A virtual network interface (NIC) will be formed for the VM for each network selected. Therefore, if the VM needs two or more NICs, you should select two or more networks accordingly.

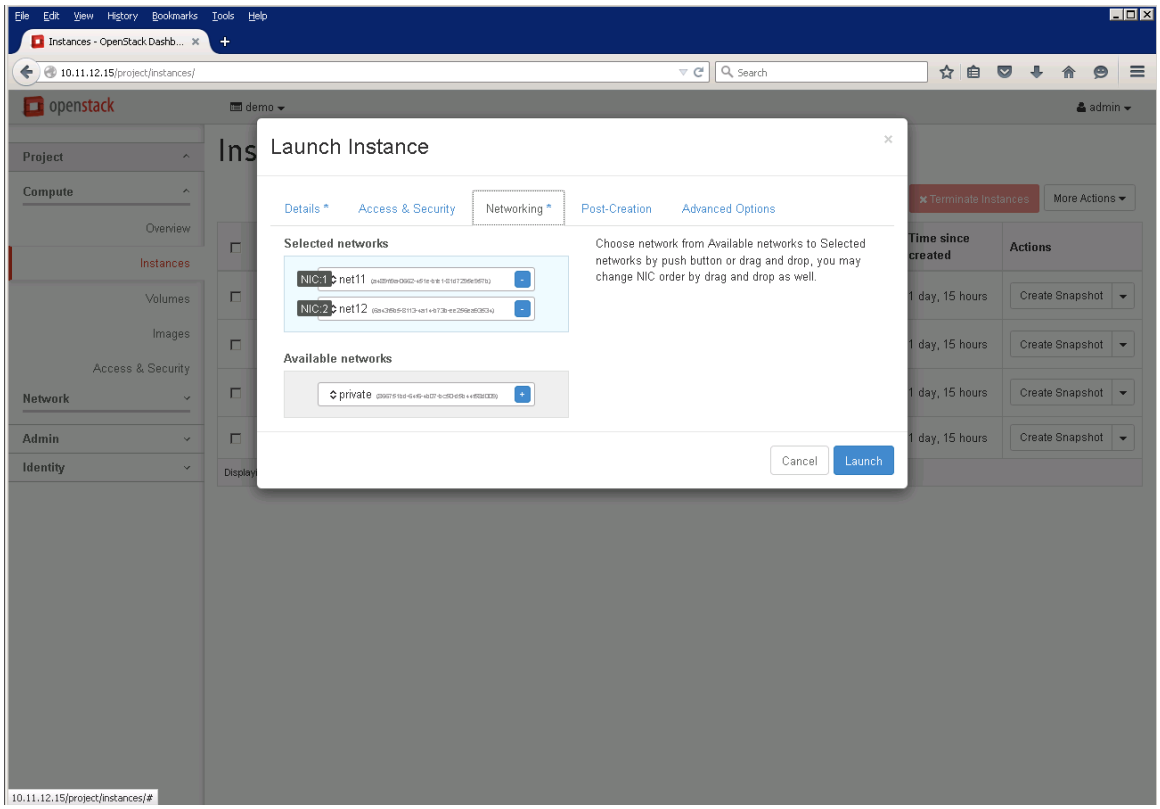

Figure 8-8 OpenStack Instances UI – Assigning Networks to an Instance

7. Click **Launch** to finish. The instance has two network interfaces, eth0 and eth1, belonging to two different networks.

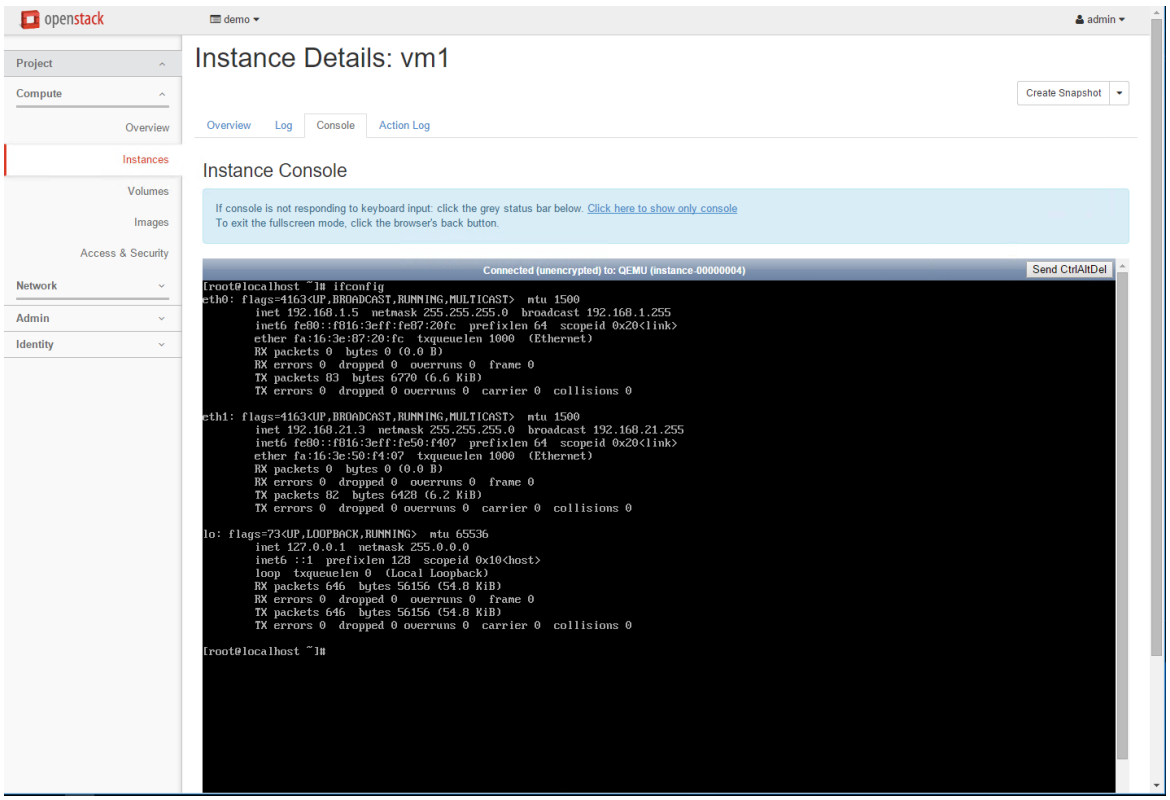

Figure 8-9 OpenStack Instances UI – Console - Displaying Network Interfaces

8. The new VM should be up and running in a minutes or so. Click the new name of the VM from the list, and then click **Console** in the top menu to access the VM.
9. Once all the 3 VMs are configured correctly with the networks, the network topology should look like in the Figure 8-10:

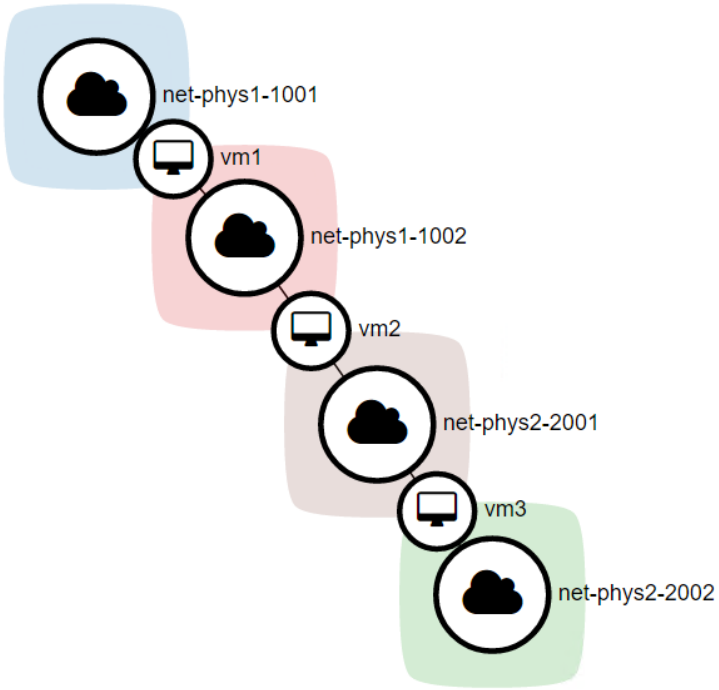

Figure 8-10 OpenStack network topology for vE-CPE setup

10. Follow the steps detailed in 8.2.4, 8.2.5, 8.2.6 to provision the VM with DPPD tool with following considerations in the `handle_l2fwd.cfg` file:
    - For unidirectional flow the "dst mac=" value should be the mac address of the next hop for the packets to reach. The Figure 9.3.1 illustrates an example as to how to configure the DPPD configuration files in each of the VNF. Following the unidirectional flow indicated by blue arrow, next hop for packets from eth1 interface of VNF1 is eth0 interface of VNF2 and so on. To detail the example, configure the handle_l2fwd.cfg of each VNF in following way:
        - In VNF1, dst_mac=00:00:00:00:00:20
        - In VNF2, dst_mac=00:00:00:00:00:30
        - In VNF3, dst_mac=00:22:00:00:00:10
    - For bidirectional flow the "dst mac=" value should be the mac address of the next hop for the packets to reach. The Figure 8-11 illustrates an example as to how to configure the DPPD configuration files in each of the VNF. The green and blue arrows indicate two different flows there by providing a reference of what next hop mac address should be for each interface of VNFs:
        - In VNF1: for task 0 dst_mac=00:00:00:00:00:20, for task 1 dst_mac=00:11:00:00:00:10
        - In VNF1: for task 0 dst_mac=00:00:00:00:00:30 , for task 1 dst_mac=00:00:00:00:00:11
        - In VNF1: for task 0 dst_mac=00:22:00:00:00:10 , for task 1 dst_mac=00:00:00:00:00:21

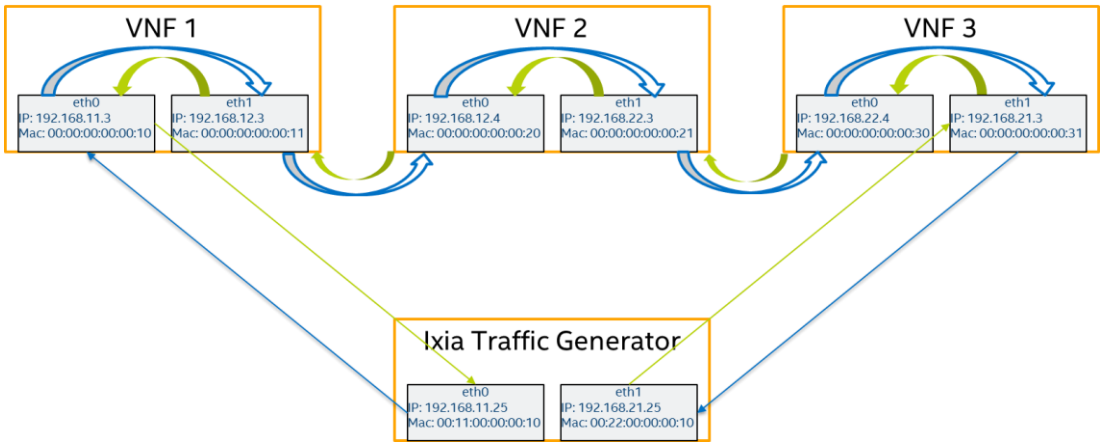

Figure 8-11 Port Mapping of VNFs for DPPD Configuration

Note: For benchmarking purpose, ensure to follow the steps listed in 9.2 Educating OvS of TCP/IP-less DPDK Based VNFs before starting DPPD application.

11. Execute the DPPD application (from 8.2.5 or 8.2.6) as a L2 forwarding VNF

With the correct setup, as the traffic from packet generator reaches the virtio ports of the VM, the packet counters in DPPD application will be live indicate Rx, Tx and packet drop statistics.

9.0 Performance Tuning Best Known Methods

9.1 OvS Configuration by OpenStack

Using the networking-ovs-dpdk ML2 plugin OpenStack takes care of configuring and setting up the bridges and their internal routing. For a first timer it is not trivial to understand the packet path with the OpenStack in the mix. The only way for the user can configure the bridges is before deploying OpenStack using `prepare_stack.sh` script, by editing the `local.conf` DevStack configuration file.

For the vE-CPE use case we use the following VLAN overlay related settings in the `local.conf` file:

```
Q_ML2_PLUGIN_MECHANISM_DRIVERS=ovsdpdk
Q_ML2_PLUGIN_TYPE_DRIVERS=vxlan,vlan,flat,local
Q_ML2_TENANT_NETWORK_TYPE=vlan
ENABLE_TENANT_TUNNELS=False
ENABLE_TENANT_VLANS=True
OVS_BRIDGE_MAPPINGS=physnet1:br-eth0,physnet2:br-eth1
ML2_VLAN_RANGES=physnet1:1000:1010,physnet2:2000:2010
```

Once the DevStack installation is complete, OvS bridge and their corresponding port setup will be complete as shown in Figure 9-1:

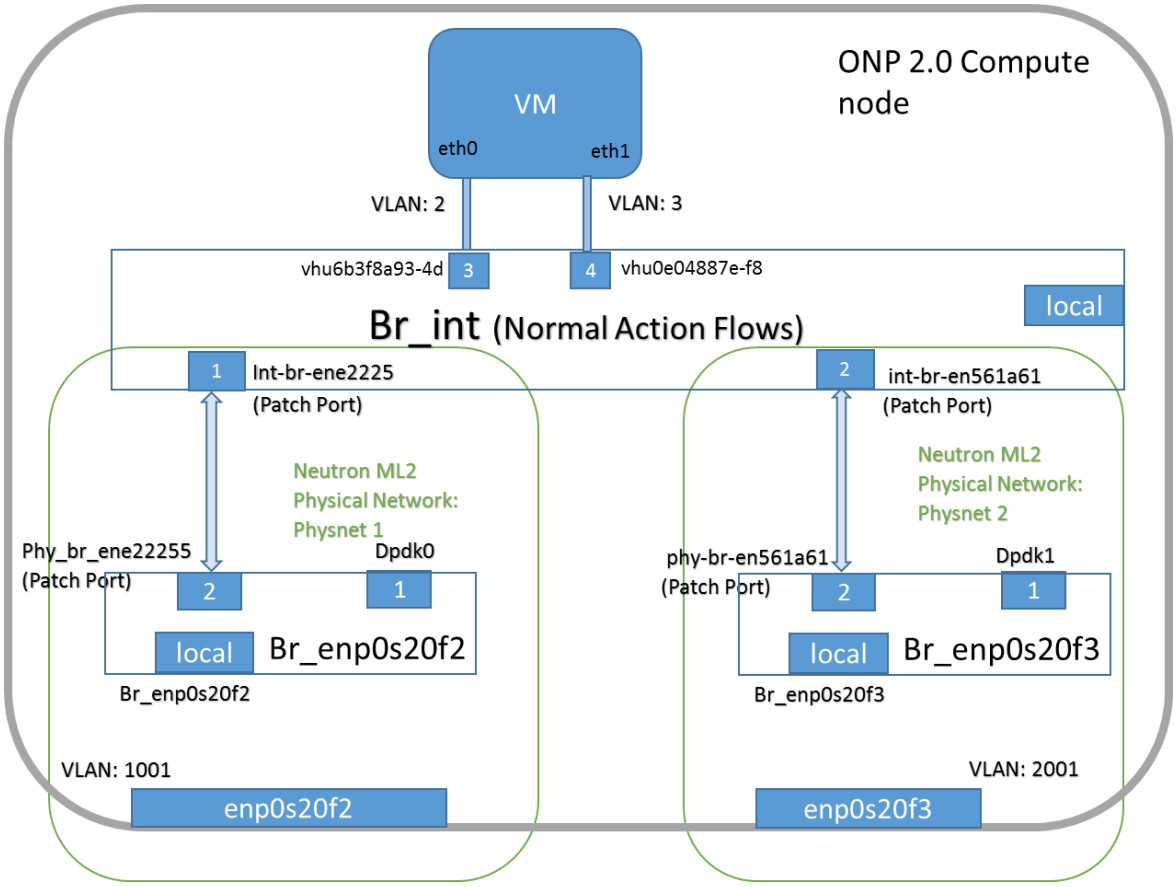

Figure 9-1 OvS bridge configuration by OpenStack

The following commands will provide the bridge details and their corresponding ports.

Note: The sample output shown here is for a compute node with one VNF.

To display the OVS bridges:

```
#ovs-vsctl show
ba61aacb-2fb2-4b2d-8b61-37b6d75d7cf2
    Bridge "br-enp0s20f2"
        Port "phy-br-ene22255"
            Interface "phy-br-ene22255"
                type: patch
                options: {peer="int-br-ene22255"}
        Port "br-enp0s20f2"
            Interface "br-enp0s20f2"
                type: internal
        Port "dpdk0"
            Interface "dpdk0"
                type: dpdk
    Bridge br-int
        fail_mode: secure
        Port "int-br-ene22255"
            Interface "int-br-ene22255"
                type: patch
                options: {peer="phy-br-ene22255"}
        Port "int-br-en561a61"
            Interface "int-br-en561a61"
                type: patch
                options: {peer="phy-br-en561a61"}
        Port br-int
            Interface br-int
                type: internal
    Bridge "br-enp0s20f3"
        Port "br-enp0s20f3"
            Interface "br-enp0s20f3"
                type: internal
        Port "phy-br-en561a61"
            Interface "phy-br-en561a61"
                type: patch
                options: {peer="int-br-en561a61"}
        Port "dpdk1"
            Interface "dpdk1"
                type: dpdk
```

To display the openflow port details on each bridge:

```
#ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000aebee777e849
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(int-br-en561a61): addr:12:92:27:a6:01:dc
```

```
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 2(int-br-ene22255): addr:ea:42:e1:d6:76:8b
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 3(vhu6b3f8a93-4d): addr:00:00:00:00:00:00
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
 4(vhu0e04887e-f8): addr:00:00:00:00:00:00
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-int): addr:ae:be:e7:77:e8:49
     config:     PORT_DOWN
     state:      LINK_DOWN
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0

#ovs-ofctl show br-enp0s20f2
OFPT_FEATURES_REPLY (xid=0x2): dpid:00000cc47a6a6fb2
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:0c:c4:7a:6a:6f:b2
     config:     0
     state:      0
     current:    1GB-FD
     speed: 1000 Mbps now, 0 Mbps max
 2(phy-br-ene22255): addr:fe:e5:2f:f3:b4:24
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-enp0s20f2): addr:0c:c4:7a:6a:6f:b2
     config:     PORT_DOWN
     state:      LINK_DOWN
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0

# ovs-ofctl show br-enp0s20f3
OFPT_FEATURES_REPLY (xid=0x2): dpid:00000cc47a6a6fb3
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(dpdk1): addr:0c:c4:7a:6a:6f:b3
     config:     0
     state:      0
     current:    1GB-FD
     speed: 1000 Mbps now, 0 Mbps max
 2(phy-br-en561a61): addr:06:8c:db:6a:a8:14
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-enp0s20f3): addr:0c:c4:7a:6a:6f:b3
     config:     PORT_DOWN
     state:      LINK_DOWN
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

9.2 Educating OvS of TCP/IP-less DPDK Based VNFs

OpenStack Neutron agent configures the integration bridge, br-int, of OvS to use normal action flow rules to have the switch learn the port information of the VNFs being deployed. The integration bridge is the interface bridge between all the VNFs deployed. Normal action flow rules are important in a NFV based deployment since the VNFs can be deployed or deleted at any time by the VNFI manager.

However, this creates a problem with VNFs that have DPDK based networking without TCP/IP networking stack. The way integration bridge learns about the ports available to send or receive traffic is through ICMP packets between the VNFs (east-west traffic) or from tenant port of the hypervisor to the VNFs (north-south traffic). With DPDK based VNFs there are no ARP responses for the switch to learn about the VNF's port information. Thus for any packets destined for the DPDK based VNFs switch will start flooding across all the ports on tenant bridge and integration bridge, ultimately causing the packets to be dropped.

To overcome this limitation we ensure that the vSwitch learns the port information of the VNF before instantiating the DPPD forwarding application. Follow the steps below to verify the vSwitch has learnt the port information of the VNFs:

1. Configure the Ixia source port IP address with the same subnet as the VNF port that will be receiving the traffic from source Ixia port. For example, from Figure 8-11, eth0 of Ixia and eth0 of VNF 1 are on same subnet
2. Configure the Ixia destination port IP address with the same subnet as the VNF port that will be sending the traffic to designation Ixia port. For example, from Figure 8-11, eth1 of Ixia and eth1 of VNF 3 are on same subnet.
3. Send a ping from receiving port of VNF1 to its corresponding source port of Ixia, eth0 of VNF1 to eth0 of Ixia.
4. Send a ping from destination port of Ixia to its corresponding transmitting port of VNF3, eth1 of Ixia to eth1 of VNF3.
5. This way we ensure the pings have resolved in a reverse direction than the actual traffic flow, example reverse direction of flow indicated in blue
6. Ping the ports across the 3 VNFs that are on same subnet. For example, from Figure 8-11:
   - ping 192.168.12.4 from 192.168.12.3
   - ping 192.168.22.4 from 192.168.22.3
7. Verify that the vSwitch has learnt about ports of all the VNFs on the integration bridge, br-int :

```
# cd /opt/stack/ovs/utilities/
# ./ovs-appctl --target=/var/run/openvswitch/ovs-vswitchd.6880.ctl \
          fdb/show br-int
```

8. Verify that the tenant port bridges have learnt about the corresponding VNF port and Ixia ports with this command:

```
# cd /opt/stack/ovs/utilities/
# ./ovs-appctl --target=/var/run/openvswitch/ovs-vswitchd.6880.ctl \
          fdb/show br-enp0s20f2
```

9. Execute DPPD after start the traffic from Ixia packet generator.
10. Following the commands detailed in 9.3 and 9.4 make sure that there are no packet drops across all the bridges thus verifying that there is no flooding across the vSwitch.

The time for each bridge after which the learnt entries in the vSwitch will expire is 300 seconds by default. User can increase this limit to a maximum of 3600 seconds to enable more time for setup and deployment. Use this command to increase the expiration limit:

```
# cd /opt/stack/ovs/
# ./utilities/ovs-vsctl set bridge <bridge name> \
          other_config:mac-aging-time=3600
```

9.3 OvS Control Path Configuration by OpenStack

To optimize the packet flow across the host platform and between the VNFs it is important to understand control path of OvS. Tools like `ovs-ofctl` provides the control path configuration details. With OpenStack being the orchestrator, the default packet path across of a unidirectional flow in the control path ports is described in Figure 9-2:

Note: For simplicity, Intel® ONP with only 1 VM deployment details are used. The ports `enp0s20f2` and `enp0s20f3` are the two tenant ports used in this example.

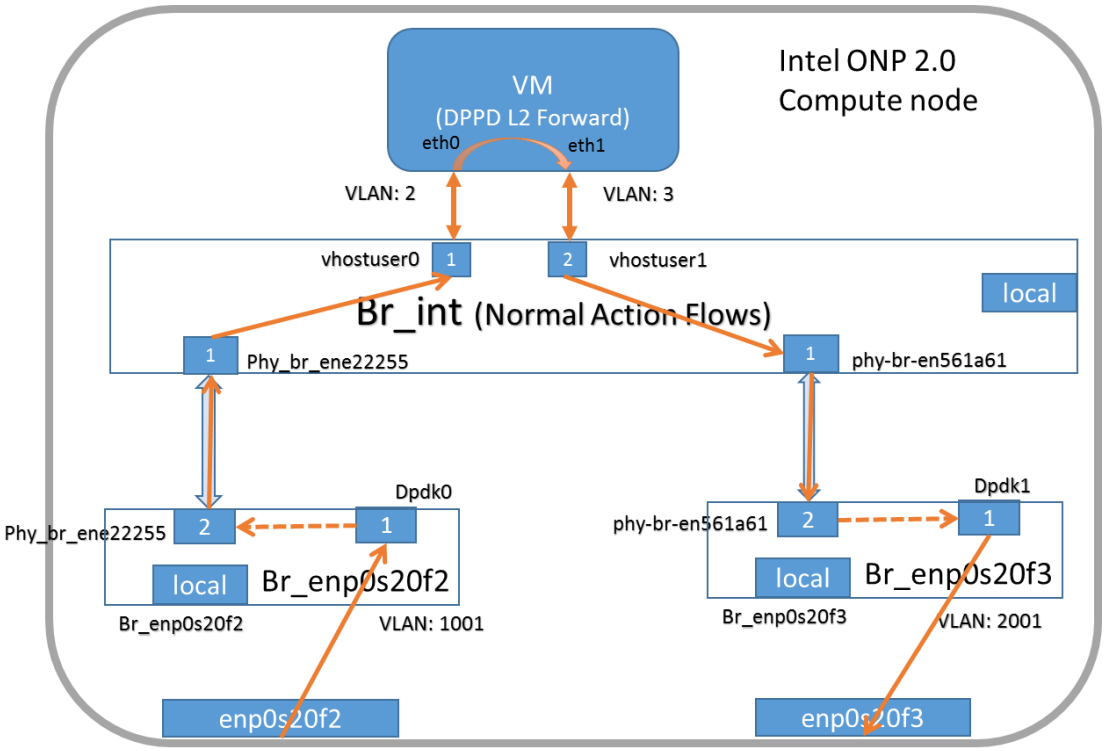

Figure 9-2 OvS Control Path packet flow

The OpenFlow port numbers of all the bridges in the control path, indicated in Figure 9-2, can be displayed using:

```
# ovs-ofctl dump-flows show br-enp0s20f2
```

To display the OpenFlow control flows on each bridge:

```
# ovs-ofctl dump-flows br-enp0s20f2
```

To display the packet statistics of every port in control path on each bridge:

```
# ovs-ofctl dump-ports br-enp0s20f2
```

It is important to track if there is any packet loss or flooding on each of the bridges configured. Using `ovs-ofctl` one can determine number of packets dropped in the bridges. By doing a `watch` on the `ovs-ofctl` command, use can know packet receive, transmit and drop statistics for ever second granularity. An example is given below:

```
# watch -d ovs-ofctl dump-flows br-enp0s20f2
# watch -d ovs-ofctl dump-ports br-enp0s20f2
```

9.4 OvS Data Path Configuration by OpenStack

To configure and optimize port level specifics across the OvS bridges it is important to understand data path of OvS. Tool like `ovs-ofctl` provides the control path configuration details. With OpenStack being the orchestrator, the default packet path across of a unidirectional flow in the control path ports is depicted in Figure 9-3:

Note: For simplicity, Intel® ONP with only 1 VM deployment details are used. The ports `enp0s20f2` and `enp0s20f3` are the two tenant ports used in this example.

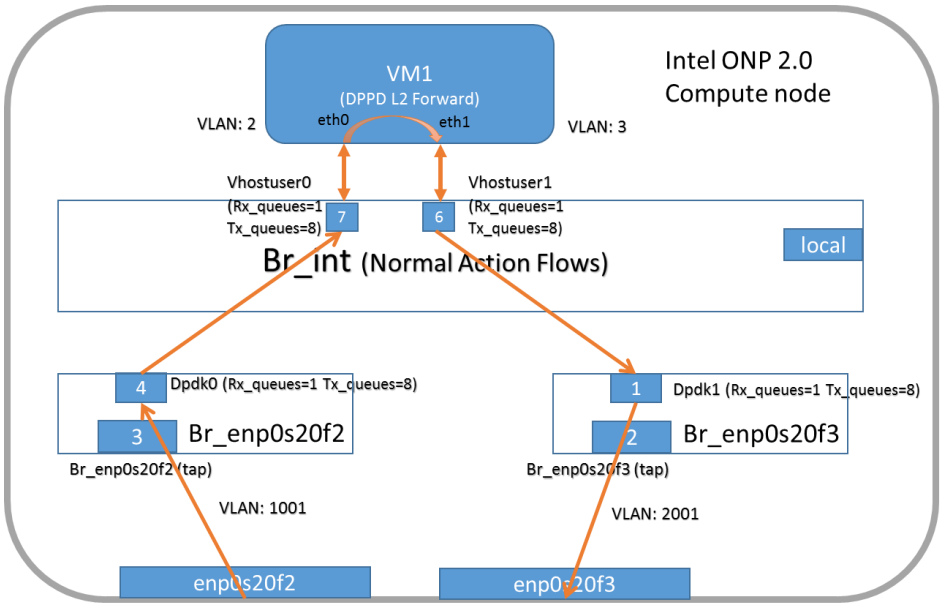

Figure 9-3 OvS data path Packet Flow

The OpenFlow port numbers of all the bridges in the data path, indicated in in Figure 9-3, can be displayed using `dpctl` command as below. It also provides the number of DPDK Rx and Tx queues configured for each port. The default number of queues for each port is the number of cores on the host hypervisor.

```
# cd /opt/stack/ovs/utilities/
# ./ovsappctl --target =/var/run/openvswitch/ovs-vswitchd.6880.ctl dpctl/show
```

To display OpenFlow data path flows across the host:

```
# cd /opt/stack/ovs/utilities/
# ./ovs-appctl --target=/var/run/openvswitch/ovs-vswitchd.6880.ctl \
        dpctl/dump-flows
```

The flow details provide how OpenStack Neutron added flow rules to modify the packets with user defined VLAN tags to OpenStack specific VLAN tags and vice versa. These level of details will help the user debug and optimize overall system performance.

To display the statistics in the data path for individual bridge:

```
# cd /opt/stack/ovs/utilities/
# ./ovs-appctl --target=/var/run/openvswitch/ovs-vswitchd.6880.ctl \
          dpif/dump-flows br-enp0s20f2
```

References

Title	Reference
01.org: Intel® Open Network Platform	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: Intel® ONP 2.0 Reference Architecture Guide	https://01.org/packet-processing/intel%C2%AE-onp-servers
01.org: Intel® ONP Release 2.0 Scripts	https://01.org/packet-processing/intel%C2%AE-onp-servers
DPDK 2.0.0 l3fwd Scalar vs Vector 10G SuperMicro Intel® Atom™ processor C2758 2400MHZ DDR3 1600MHz	http://cat.intel.com/LaunchLink.aspx?LinkID=3230
End-to-End Network Performance Rules & Objectives for the Interconnection of NGN	http://www.niccstandards.org.uk/files/current/ND1704v2_1_1.pdf?type=pdf
ETSI NFV Use Cases	http://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf
IEEE 802.3	http://standards.ieee.org/about/get/802/802.3.html
Intel® Atom™ Processor C2758	http://ark.intel.com/products/77988/Intel-Atom-Processor-C2758-4M-Cache-2_40-GHz
ITU-T Recommendation G.1020 Performance parameter definitions for quality of speech and other voiceband applications utilizing IP networks	https://www.itu.int/rec/T-REC-G.1020-200311-S/en
ITU-T Recommendation G.114 One-way transmission time	https://www.itu.int/rec/T-REC-G.114/en
ITU-T Recommendation I.356 B-ISDN ATM layer cell transfer performance	http://www.itu.int/rec/T-REC-I.356/en
ITU-T Recommendation Y.1540 Internet protocol data communication service - IP packet transfer and availability performance parameters	https://www.itu.int/rec/T-REC-Y.1540/en
IxNetwork	http://www.ixiacom.com/products/ixnetwork
RFC 1242 Benchmarking Terminology for Network Interconnection Devices	https://tools.ietf.org/html/rfc1242
RFC 2544 Benchmarking Methodology for Network Interconnect Devices	https://tools.ietf.org/html/rfc2544

Title	Reference
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 A1SRi-2758F Motherboard	http://www.supermicro.com/products/motherboard/Atom/X10/A1SRi-2758F.cfm
SuperMicro SuperServer 5018A-FTN4	http://www.supermicro.com/products/system/1U/5018/SYS-5018A-FTN4.cfm
Optimizing Infrastructure for Workloads in Openstack-Based Public Cloud Services	https://software.intel.com/sites/default/files/Optimizing%20Infrastructure%20for%20Workloads%20in%20OpenStack-based%20Cloud%20Services.pdf

Acronyms and Abbreviations

Abbreviation	Description
ARP	Address Resolution Protocol
BMWG	Benchmark Working Group
CPE	Customer Premises Equipment
CPU	Central Processing Unit
DPDK	Data Plane Development Kit
DPPD	Intel® Data Plane Performance Demonstrators
DPI	Deep Packet Inspection
DUT	Device-Under-Test
EFM	Ethernet in the First Mile
EMC	Exact Match Cache
ETSI	European Telecommunications Standards Institute
GbE	Gigabit Ethernet
GRUB	GRand Unified Bootloader
I/O	Input/Output
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IMIX	Internet Mix
IPDV	Inter-Packet Delay Variation
IPv4	Internet Protocol version 4
IPS	Intrusion Prevention System
IRQ	Interruption Request
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
KVM	Kernel-based Virtual Machine

Abbreviation	Description
LAN	Local Area Network
MAC	Media Access Control
MPLS	Multiprotocol Label Switching
NFV	Network Functions Virtualization
NIC	Network Interface Card
NUMA	Non-Uniform Memory Access
ONP	Intel® Open Network Platform
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
PROX	Packet pROcessing eXecution engine
PSTN	Public Switched Telephone Network
QEMU	Quick Emulator
RFC	Request for Comments
SBC	Session Border Controller
SDN	Software-Defined Networking
SELinux	Security-Enhanced Linux
SLA	Service-Level Agreement
TLB	Translation Lookaside Buffer
UEFI	Unified Extensible Firmware Interface
VDSL	Very-high-bit-rate Digital Subscriber Line

Abbreviation	Description
vE-CPE	Virtual Enterprise Customer Premises Equipment
vhost	Virtual Host
VIM	Virtual Infrastructure Manager
VLAN	Virtual LAN
VM	Virtual Machine
VNF	Virtualized Network Function
VNFaaS	VNF as a Service
VoIP	Voice over IP
VPN	Virtual Private Network
VSAT	Very-Small-Aperture Terminal
VTEP	VXLAN Tunnel End Point
VXLAN	Virtual eXtensible LAN
WAN	Wide Area Network
xDSL	x Digital Subscriber Line

Legal Information

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

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

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

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

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

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

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

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

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