Intel® Open Network Platform Server (Release 1.3)

Benchmark Performance Test Report

Document Revision 1.3.1
June 2015

Revision History

Revision	Date	Comments
1.3.1	June 18, 2015	Minor edit to remove reference to earlier software version
1.3	June 17, 2015	Document release aligned with Intel® Open Network Platform Server 1.3, but delayed for 1.4
1.0	December 15, 2014	Initial document for release of Intel® Open Network Platform Server 1.2

Contents

Figures

Tables

1.0 Introduction

The Intel® Open Network Platform Server Benchmark Performance Test Report provides a guide for data plane performance characterization using the Intel® Open Network Platform Server (Intel ONP Server).

The primary audiences for this document are architects and engineers evaluating the Intel® Open Network Platform Server. Software ingredients of this architecture include:

- Fedora Linux*
- Intel® Data Plane Development Kit (DPDK)
- Open vSwitch (OVS)*
- OpenStack*
- OpenDayLight*

Information in this document includes instructions for using certain software patches to improve system performance.

This setup as described can help to evaluate specific platform performance requirements. The tests described are not comprehensive and do not include protocol conformance or interoperability aspects. Providing a baseline platform configuration with benchmark methodologies and performance data, using tested and documented procedures, can help architects and engineers achieve optimal system performance for developing NFV/SDN solutions.

2.0 Ingredient Specifications

2.1 Hardware Versions

Table 2–1 provides information on the platform hardware components used for our testing purposes. The "Notes" column details some of the fine tunings enabled on the hardware.

Table 2–1. Hardware Ingredients

Item	Description	Notes
Platform	Intel® Server Board S2600WT2	Intel® Server Board S2600WT2, 1 Gb Ethernet ports, 1TB HDD 3.5in SATA 7200RPM Seagate Enterprise Constellation ES.3 ST1000NM0033
Processor	Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60 GHz	Xeon DP Haswell-EP E5-2697 v3 LGA2011 2.6 GHz, 45 MB, 145 W, 14 cores
Cores	14 physical cores per CPU (per socket)	28 hyperthreaded cores per CPU (per socket) for a total of 54 cores per platform (2 sockets)
Memory	16 GB DDR4 DIMM Crucial	128 GB RAM: 8 x 16 GB 2133 Reg ECC 1.2V DDR4 RDIMM Crucial CT16G4RFD4213
NICs (Niantic)	2x Intel® 82599 10 Gigabit Ethernet controller	Niantic PCIe v2.0 Dual 10 GbE SFP+ Ports
NICs (Fortville)	X710-DA4 Quad port 10 GbE Adapter	Fortville PCIe v3.0 (8.0 GT/s) Quad Port Ethernet Intel X710DA4G1P5 SFP+
BIOS	BIOS Revision: Unknown Release Date: 08/11/2014	Intel® Virtualization Technology for Directed I/O (Intel® VT-d) enabled; hyperthreading disabled
Quick Assist Technology	Intel® QuickAssist Adapter 8950-SCCP (formerly code-named Walnut Hill) with Intel® Communications Chipset 8950 (formerly code-named Coleto Creek)	PCIe acceleration card with Intel® Communications Chipset 8950; capabilities include RSA, SSL/IPsec, wireless crypto, security crypto, and compression

Note: Tests were performed with either Fortville or Niantic NICs as indicated in the test results.

2.2 Software Versions

Table 2–2. Hardware Ingredients

Software Component	Function	Version/Configuration
Fedora 21 x86_64	Host OS	3.17.4-301.fc21.x86_64
Fedora 21 x86_64 (Fedora server)	VM	3.17.4-301.fc21.x86_64
Qemu-kvm	Virtualization Technology	qemu-kvm.x86_64 2:2.1.3-1.fc21
Data Plane Development Kit (DPDK)	Network Stack bypass	DPDK 1.8.0 git commit 6fb3161060fc894295a27f9304c56ef34492799d
Open vSwitch (OVS)	vSwitch	Open vSwitch V 2.3.90 git commit 7cc398cb8561a16ae3be5ffc687be5620981d619
DPDK-netdev[1]	vSwitch patch	This is currently an experimental feature of OVS that needs to be applied to OVS version specified above. http://openvswitch.org/pipermail/dev/2015-March/052061.html
Intel® QuickAssist Technology Driver (QAT)	Crypto Accelerator	Intel® Communications Chipset 89xx Series Software for Linux* - Intel® QuickAssist Technology Driver (L.2.2.0-30). Latest drivers posted at: https://01.org/packet-processing/intel%C2%AE-quickassist-technology-drivers-and-patches

Note: A line rate at 10 Gb/s in a flow of 64-byte packets requires processing more than 14.8 million packets per second (Mpps), but current standard version of OVS kernel pushes more like 1.1 million to 1.3 million and has a high latency. OVS with DPDK-netdev has been on GitHub since March and is available as an "experimental feature" in OVS version 2.3.

3.0 Test Cases Summary

Network throughput, latency, and jitter performance are measured for L2/L3 forwarding and use the standard test methodology RFC 2544 (refer to Appendix K.1 for details).

L3 forwarding on host and port-port switching performance using the vSwitch also serves as a verification after installing the compute node ingredients. Table 3–1 is a summary of performance test cases and represents a subset of many possible test configurations.

Table 3–1. Test Case Summary

Test Case	Description	Configuration	Workload/Metrics
Host-Only Performance			
1	L3 Forwarding on host	DPDK	• L3 Fwd (DPDK test application) • No packet modification • Throughput[1] • Latency (avg)
2	L2 Forwarding on host	DPDK	• L2 Fwd (DPDK Application) • Packet MAC modification • Throughput[1] • Latency (avg, max, min)
Virtual Switching Performance			
3	L3 Forwarding by vSwitch — throughput	OVS with DPDK-netdev[2]	• Switch L3 forward • Throughput[1] • Latency (avg)
4	L3 Forwarding by vSwitch— latency	OVS with DPDK-netdev[2]	• Switch L3 Fwd • Latency distribution (percentile ranges)[3]
5	L3 Forwarding by single VM	Passthrough	• L2 Fwd in VM • Pkt MAC modification • Throughput1 • Latency (avg)
6	L2 Forwarding by single VM	SR-IOV	• L2 Fwd in VM • Packet MAC modification • Throughput1 • Latency (avg)
7	L3 Forwarding by single VM	OVS with DPDK-netdev[2] User space vhost	• L3 table lookup in switch with port forward (test PMD app) in VM without packet modification • Throughput[1] • Latency (avg)

Table 3–1. (Cont'd)

Test Case	Description	Configuration	Workload/Metrics
8	L3 Forwarding by two VMs in series	OVS with DPDK-netdev[2] user space vhost	• L3 table lookup in switch with port foward (test pmd app) in VM without packet modification • Throughput[1] • Latency (avg)
9	L3 Forwarding by VMs in parallel	OVS with DPDK-netdev[2] user space vhost	• L3 table lookup in switch with port foward (test pmd app) in VM without packet modification • Throughput[1] • Latency (avg)

Notes:

1. Tests ran for 60 seconds per load target.
2. OVS with DPDK-netdev test cases were performed using OVS with DPDK-netdev, which is available as an "experimental feature" in OVS version 2.3. This shows the performance improvements possible today with standard OVS code when using DPDK. Refer to section 2.2, Software Versions, for details of software versions with commit IDs.
3. Tests ran 1 hour per load target.

4.0 Platform Configuration

The configuration parameters described in this section are based on experience running various tests and may not be optimal.

4.1 Linux Operating System Configuration

Table 4–1. Linux Configuration

Parameter	Enable/Disable	Explanation
Firewall	Disable	Disable
NetworkManager	Disable	As set by install
irqbalance	Disable	
ssh	Enabled	
Address Space Layout Randomization (ASLR)	Disable	
IPV4 Forwarding	Disable	
Host kernel boot parameters (regenerate grub.conf)	default_hugepagesz=1G hugepagesz=1G hugepages=16 hugepagesz=2M hugepages=2048 'intel_iommu=off' isolcpus=1,2,3,4,5,6,7,8,9…n-1 for n core systems	Isolate DPDK cores on host. 1 GB and also 2 MB hugepages
VM kernel boot parameters (regenerate grub.conf)	default_hugepagesz=1GB hugepagesz=1GB hugepages=1 hugepagesz=2M hugepages=256 isolcpus=1,2	Isolate DPDK cores on VM. 1 GB and also 2 MB hugepages

4.2 BIOS Configuration

The configuration parameters described in this section are based on experiments running the tests described and may not be optimal.

Table 4–2. BIOS Configuration

Parameter	Enable/Disable	Explanation
Power Management Settings		
CPU Power and Performance Policy	Traditional	This is the equivalent of "performance" setting.
Intel® Turbo boost	Enable	
Processor C3	Disable	Prevents low power state
Processor C6	Enable	Not relevant when C3 is disabled
Processor Configuration		
Intel® Hyper-Threading Technology (HTT)	Enable	
MLC Streamer	Enable	Hardware Prefetcher: Enabled
MLC Spatial Prefetcher	Enable	Adjacent Cache Prefetch: Enabled
DCU Data Prefetcher	Enable	DCU Streamer Prefetcher: Enabled
DCU Instruction Prefetcher	Enable	DCU IP Prefetcher: Enabled
Virtualization Configuration		
Intel® Virtualization Technology for Directed I/O (VT-d)	Enable	This is disabled by the kernel when iommu = off (default).
Memory Configuration		
Memory RAS and Performance Configuration –> NUMA Optimized	Auto	
I/O Configuration		
DDIO (Direct Cache Access)	Auto	

4.3 Core Usage for OVS

Table 4–3. Core OVS Usage with Intel DPDK

Process	Core	Linux Commands
ovs-vswitchd (all tasks except pmd)	3	OVS user space processes affinitized to core mask 0x04: `# ./vswitchd/ovs-vswitchd --dpdk -c 0x4 -n 4 --socket-mem 2048,2048 -- unix:$DB_SOCK --pidfile`
ovs-vswitchd pmd task	4	Sets the PMD core mask to 2 for CPU core 1 affinitization (sets in the OVS database and is persistent): `# ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=2`

All Linux threads for OVS vSwitch daemon can be listed with:

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

Using `taskset`, the current task thread affinity can be checked and changed to other CPU cores as needed.

5.0 Test Results

In this section, all performance testing is done using RFC 2544 test methodology with zero packet loss. The detailed test setup is documented in the appendixes.

5.1 Host Performance with DPDK

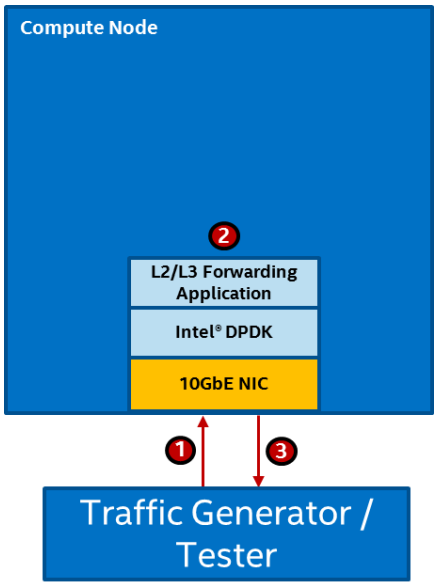

Figure 5–1. Host Performance with DPDK

Configuration:

1. No VMs are configured and no vSwitch is started or used.
2. DPDK-based port forwarding sample application (L2/L3fwd).

Data Path (Numbers Matching Red Circles):

1. The packet generator creates flows based on RFC 2544.
2. The DPDK L2 or L3 forwarding application forwards the traffic from the first physical port to the second (unidirectional). For the bidirectional test, packets flows both directions.
3. The traffic flows back to the packet generator and is measured per the RFC 2544 throughput test.

5.1.1 Test Case 1 — Host L3 Forwarding

Test Configuration Notes:

- Traffic is bidirectional (two flows) through x2 10 GbE ports of a 4 x 10 GbE Fortville NIC.
- Packets are not modified.

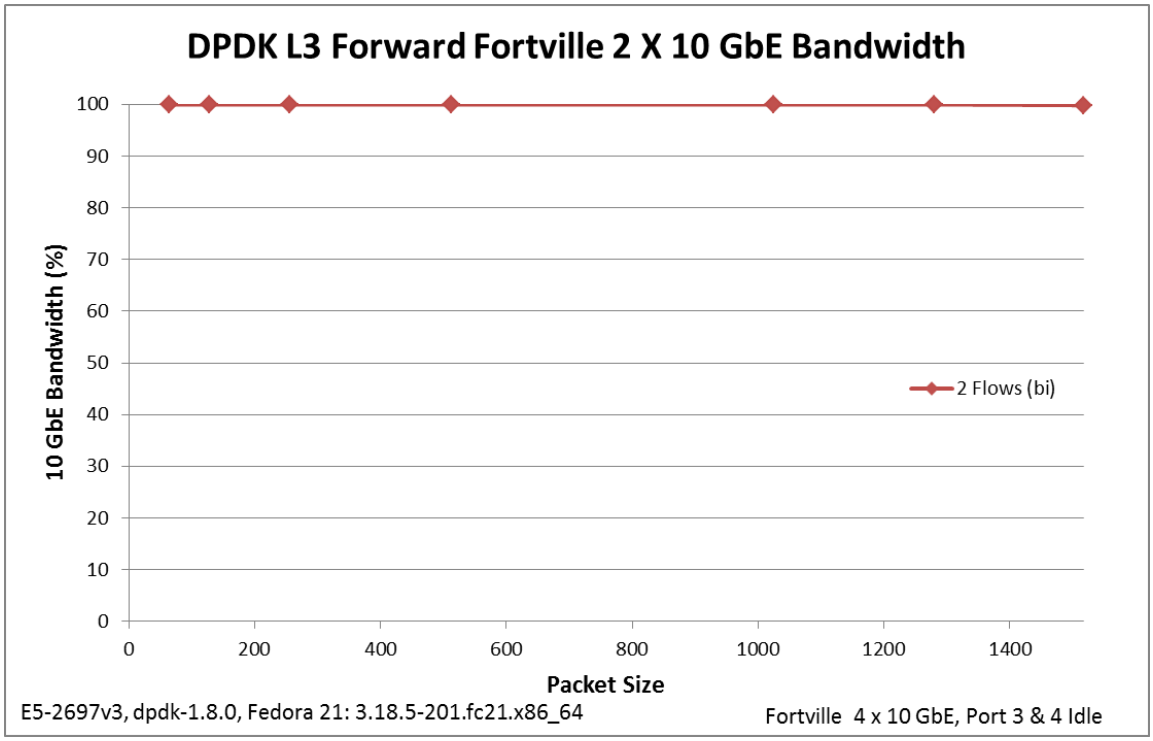

Figure 5–2. Host L3 Forwarding Performance — Throughput

Table 5–1 shows performance with bidirectional traffic. Fortville supports a full-line rate with 64B packets (zero packet loss), i.e., 29.8 Mpps using 2 x 10 GbE ports. The average latency data includes the latency of 10 GbE port (input and output), as well as the dpdk l3fwd application with a large NIC I/O queue size.

Table 5–1. Host L3 Forwarding Performance — Packet Rate, Average Latency

Packet Size (Bytes)	Packet Rate (PPS)	Average Latency (µs)	Bandwidth (% of 10 GbE)
64	29,761,873	21	100
128	16,891,868	19	100
256	9,057,956	22	100
512	4,699,241	28	100
1024	2,394,633	40	100
1280	1,923,075	46	100
1518	1,623,374	44	100

5.1.2 Test Case 2 — Host L2 Forwarding

Test Configuration Notes:

- Traffic is bidirectional (two flows) through 2 x 10 GbE ports of a 4 x 10 GbE Fortville NIC.
- Packets are modified (Ethernet header).

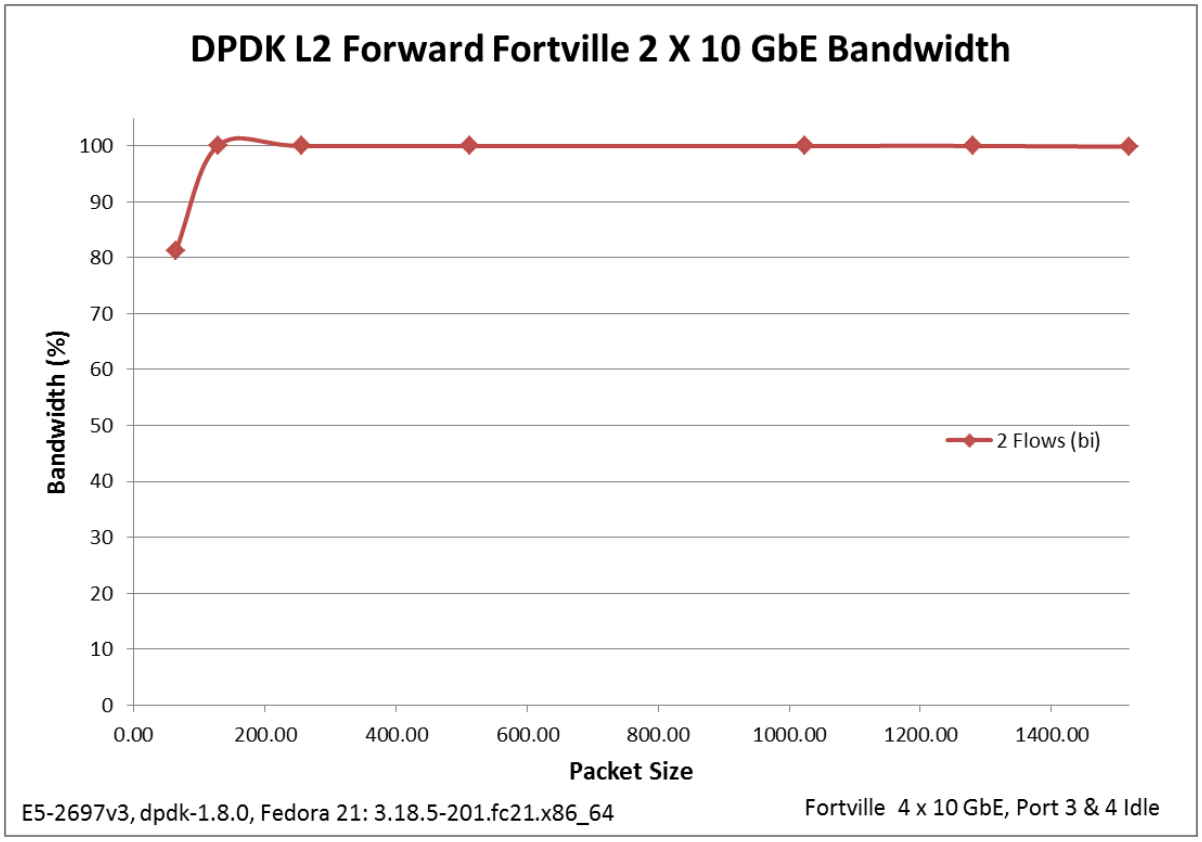

Figure 5–3. Host L2 Forwarding Performance — Throughput

Table 5–2 shows performance with bidirectional traffic. The Fortville NIC used for this test has 4 x 10 GbE ports with only 2 ports passing traffic. Results shown are measured with zero packet loss. The average latency data includes the latency of a 10 GbE port, as well as a dpdk l2fwd application with a large NIC I/O queue size.

Table 5–2. Host L2 Forwarding Performance — Packet Rate, Average Latency

Packet Size (Bytes)	Packet Rate (PPS)	Average Latency (µs)	Bandwidth (% of 10 GbE)
64	24,187,844	8	81
128	16,891,865	33	100
256	9,057,956	33	100
512	4,699,241	35	100
1024	2,394,633	44	100
1280	1,923,074	48	100
1518	1,623,375	45	100

5.2 Virtual Switching Performance

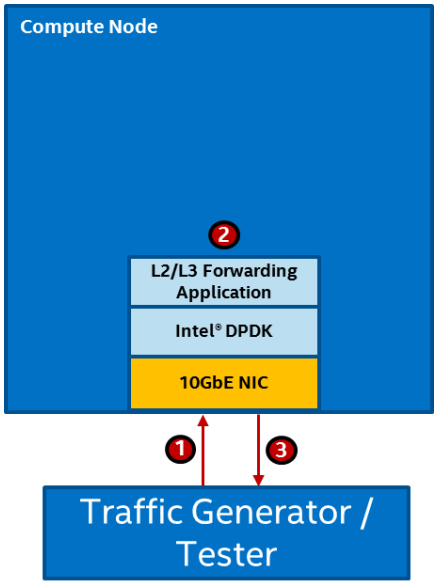

Figure 5–4. Virtual Switching Performance

Configuration:

1. No VMs are configured (i.e., PHY-to-PHY).
2. OVS with DPDK-netdev, port-to-port.

Data Path (Numbers Matching Red Circles):

1. The packet generator creates flows based on RFC 2544.
2. OVS network stack forwards the traffic from the first physical port to the second.
3. The traffic flows back to the packet generator.

5.2.1 Test Case 3 — OVS with DPDK-netdev L3 Forwarding (Throughput)

Test Configuration Notes:

- Traffic is bidirectional, except for one flow case (unidirectional). The number of flows in Figure 5–5 represents the total flows in both directions.
- Packets are not modified.
- These tests are "PHY-PHY" (physical port to physical port without a VM).
- OVS poll-mode driver (PMD) was tested with 1, 2, and 4 threads handling 10 GbE interfaces.

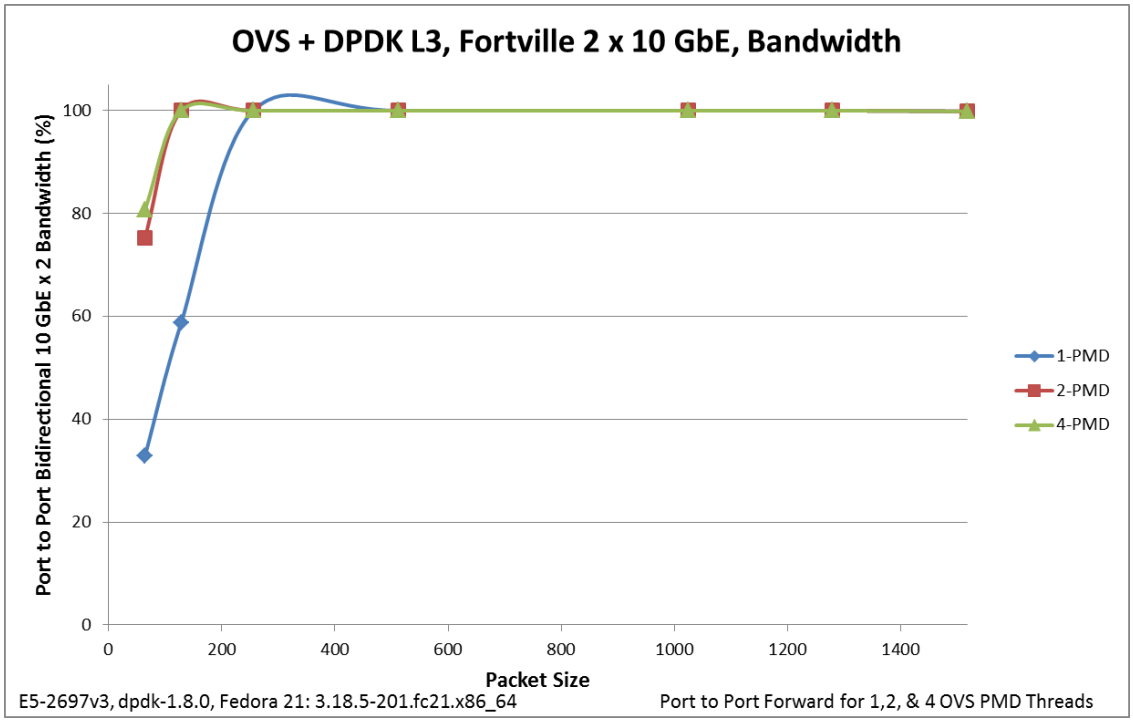

Figure 5–5. OVS with DPDK-netdev Switching Performance — Throughput

The average latency data includes the latency of 10 GbE port as well as the OVS application.

Table 5–3. OVS with DPDK L3 forwarding Performance — 2 PMD Threads

Packet Size (Bytes)	Packet Rate (PPS)	Average Latency (µs)*	Bandwidth (% of 10 GbE)
64	22,416,690	7	75
128	16,891,882	47	100
256	9,057,968	30	100
512	4,699,246	26	100
1024	2,394,635	34	100
1280	1,923,075	49	100
1518	1,623,376	8	100

Note: Latency is measured by the traffic generator on a per packet basis.

5.2.2 Test Case 4 — OVS with DPDK-netdev L3 Forwarding (Latency)

Test Configuration Notes:

- Traffic is bidirectional.
- Packets are not modified.
- Tests run for 1 hour per target load.
- Only 64-byte results are shown in Figure 5–6.
- These tests are "PHY-PHY" (physical port to physical port without a VM).

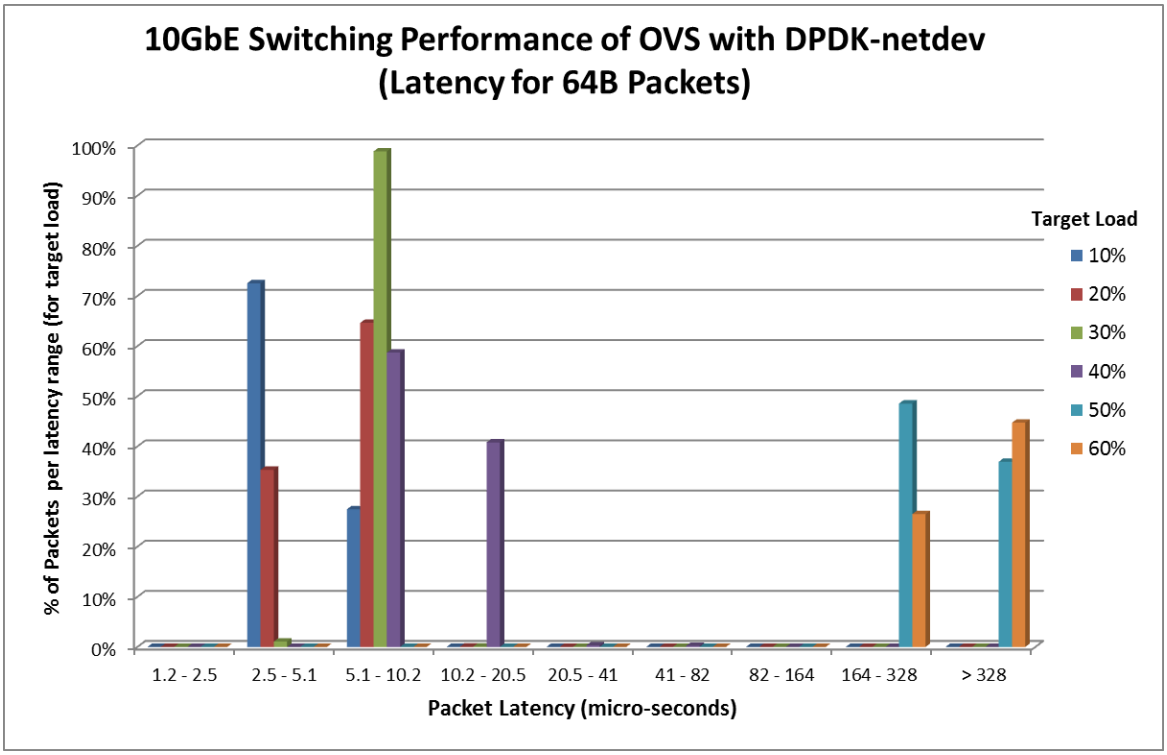

Figure 5–6. Host L2 Forwarding Performance — Latency Distribution for 64B Packets

Figure 5–6 shows the latency distribution of packets for a range of target loads. For example, with a target load of 10% (1 Gb/s) most packets have a latency of between 2.5 and 5.1 µs with an overall average of 4.844 µs (average is not displayed in the plot). Whereas at a target load of 40% (5 Gb/s), most packets have a latency of between 5.1 and 10.2 µs with an overall average of 10.144 µs (average is not displayed in the plot).

5.3 VM Throughput Performance without a vSwitch

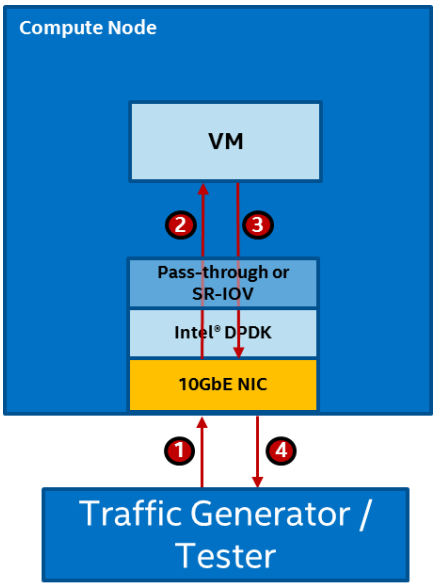

Figure 5–7. VM Throughput Performance without a vSwitch

Configuration (Done Manually):

1. One VM is brought up and connected to the NIC with PCI passthrough or SR-IOV.
2. IP addresses of the VM are configured.

Data Path (Numbers Matching Red Circles):

1. The packet generator creates flows based on RFC 2544.
2. Flow arrived at the first vPort of the VM.
3. The VM receives the flows and forwards them out through its second vPort.
4. The flow is sent back to the packet generator.

5.3.1 Test Case 5 — VM L2/L3 Forwarding with PCI Passthrough

Test Configuration Notes for L3 Forwarding:

- Niantic NIC is used.
- Traffic is bidirectional.
- Packets are not modified.

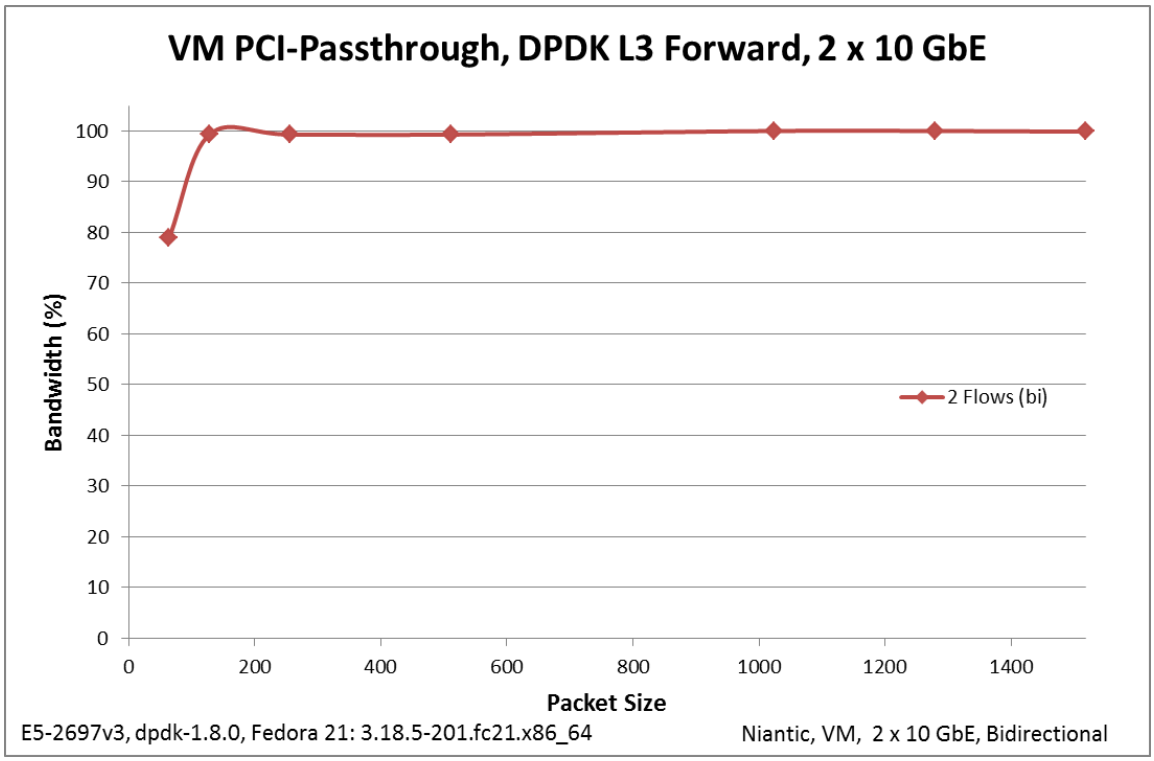

Figure 5–8. VM L3 Forwarding Performance with PCI Passthrough — Throughput

Table 5–4. VM Niantic PCI Passthrough DPDK L3 Forwarding Performance — 2 PMD Threads

Packet Size (Bytes)	Packet Rate (PPS)	Average Latency (µs)	Bandwidth (% of 10 GbE)
64	23,503,781	11	79
128	16,773,716	15	99
256	8,994,608	15	99
512	4,666,374	21	99
1024	2,394,564	924	100
1280	1,923,020	927	100
1518	1,623,374	48	100

Test Configuration Notes for L2 Forwarding:

- Niantic NIC is used.
- Traffic is bidirectional.
- Packets are modified (Ethernet header).

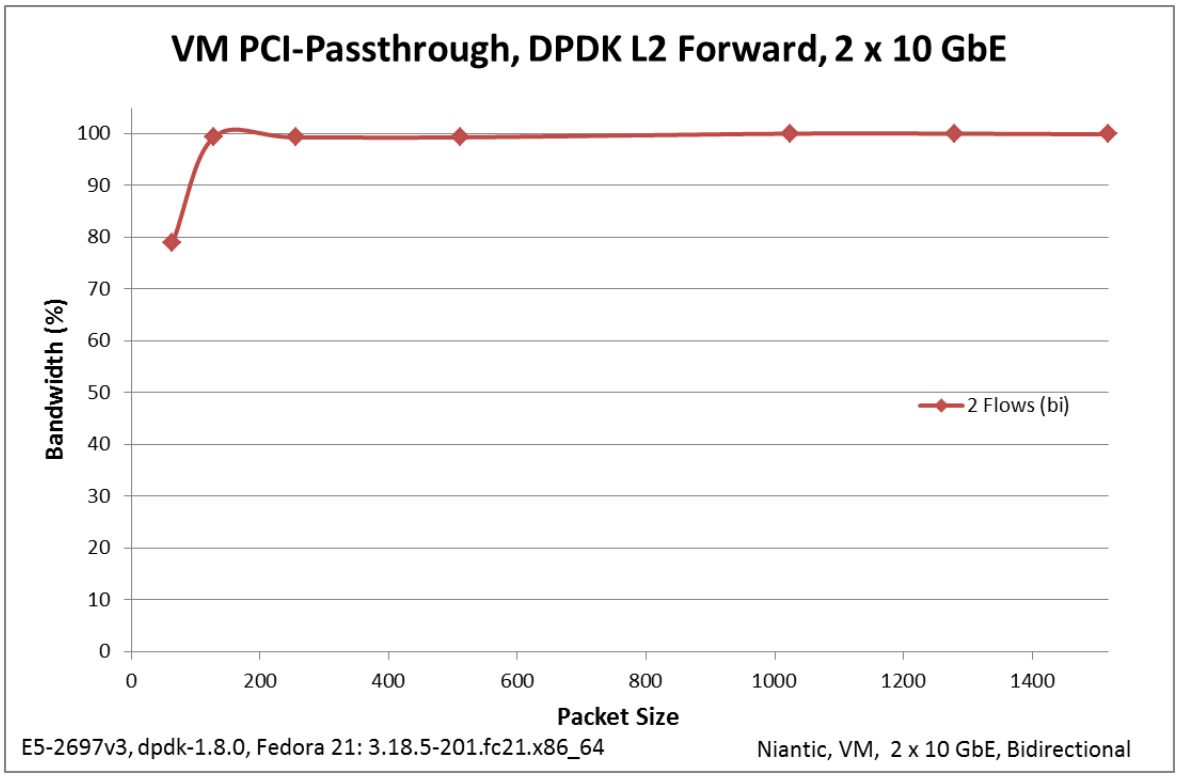

Figure 5–9. VM L2 Forwarding Performance with PCI Passthrough — Throughput

Table 5–5. VM Niantic PCI Passthrough DPDK L3 Forwarding Performance — 2 PMD Threads

Packet Size (Bytes)	Packet Rate (PPS)	Average Latency (µs)	Bandwidth (% 2x10 GbE)
64	23,503,769	11	79
128	16,773,726	15	99
256	8,994,603	15	99
512	4,666,375	21	99
1024	2,394,565	925	100
1280	1,923,020	932	100
1518	1,623,375	48	100

5.3.2 Test Case 6 — DPDK L3 Forwarding with SR-IOV

Test Configuration Notes for L3 Forwarding:

- Niantic NIC is used.
- Traffic is bidirectional.
- Packets are not modified.

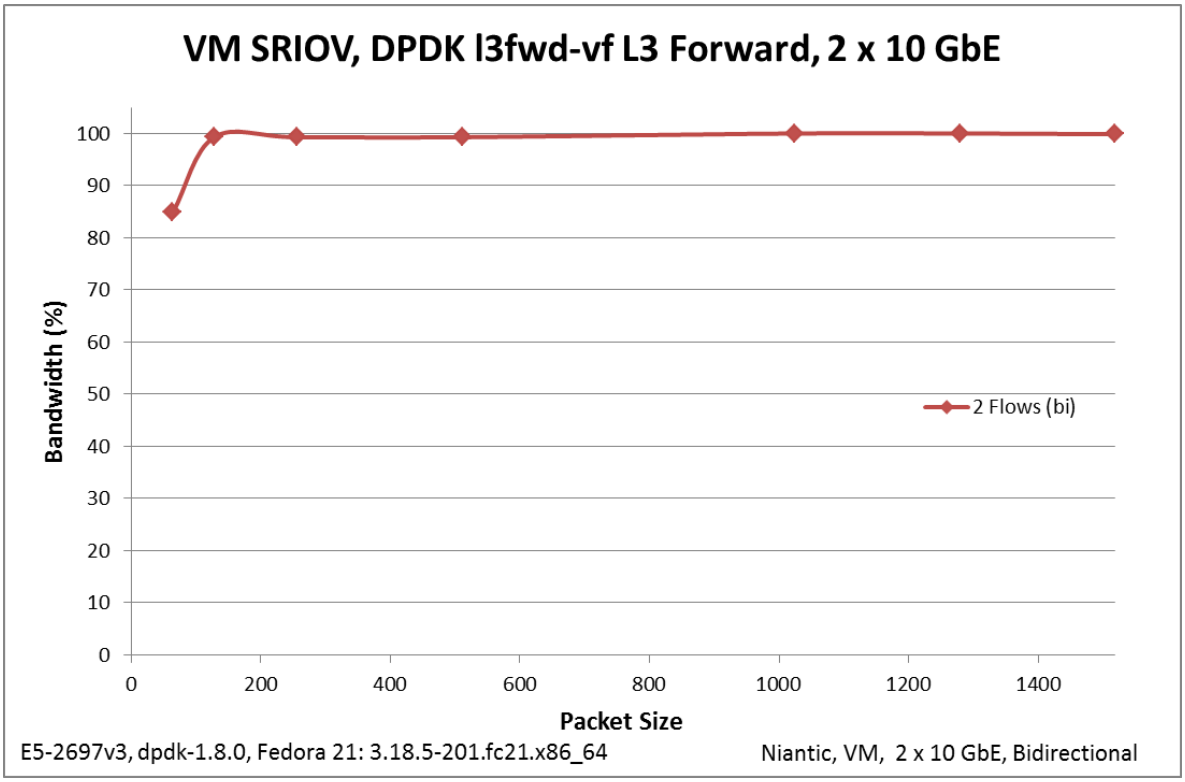

Figure 5–10. VM Niantic SR-IOV L3 Forwarding Performance — Throughput

Table 5–6. VM Niantic SR-IOV DPDK L3 Forwarding Performance — 2 PMD Threads

Packet Size (Bytes)	Packet Rate (PPS)	Average Latency (µs)	Bandwidth (% 2x10 GbE)
64	25,246,456	12	84
128	16,773,726	15	99
256	8,994,603	16	99
512	4,666,375	22	99
1024	2,394,568	890	100
1280	1,923,023	887	100
1518	1,623,375	49	100

5.4 VM Throughput Performance with vSwitch

5.4.1 Test Case 7 — Throughput Performance with One VM (OVS with DPDK-netdev, L3 Forwarding)

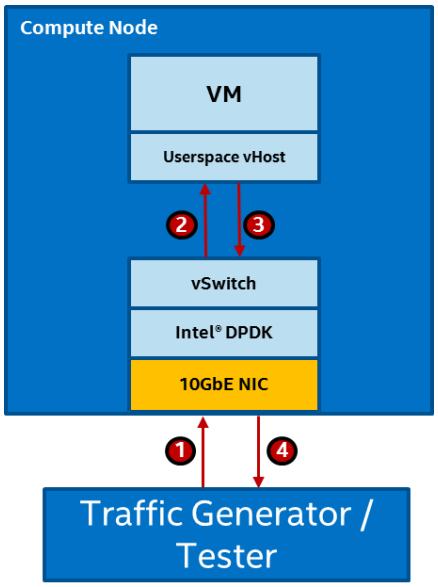

Figure 5–11. VM Throughput Performance with vSwitch — One VM

Configuration (Done Manually):

1. One VM gets brought up and connected to the vSwitch.
2. IP address of the VM gets configured.
3. Flows get programmed to the vSwitch.

Data Path (Numbers Matching Red Circles):

1. The packet generator creates flows based on RFC 2544.
2. The vSwitch forwards the flows to the first vPort of the VM.
3. The VM receives the flows and forwards them out through its second vPort.
4. The vSwitch forwards the flows back to the packet generator.

Test Configuration Notes:

- Traffic is bidirectional.
- Packets are not modified.

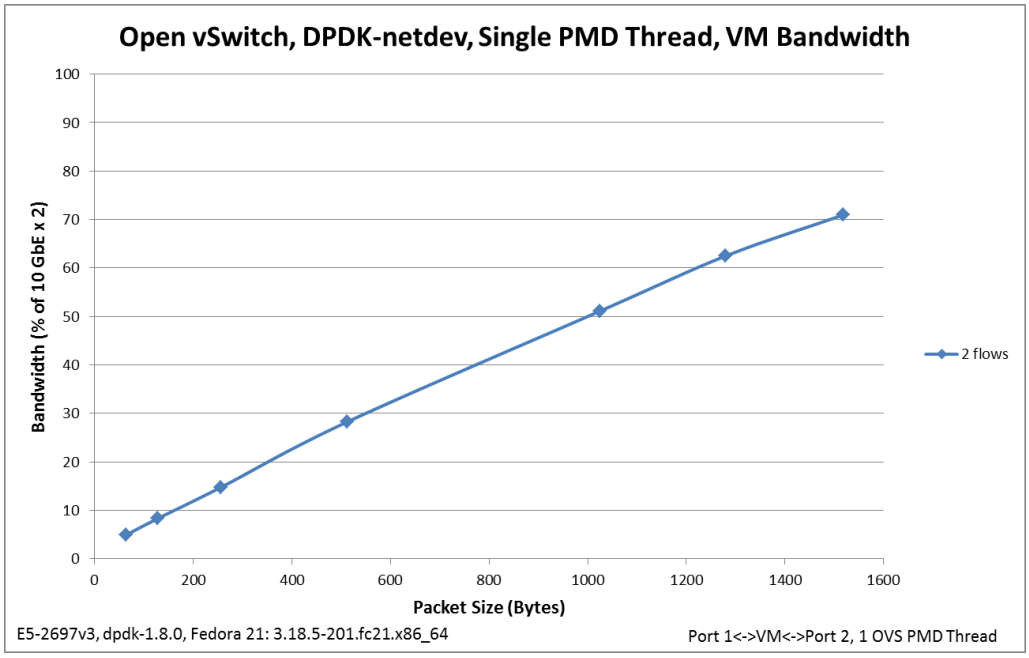

Figure 5–12. VM Throughput Performance — One VM (OVS with DPDK-netdev)

5.4.2 Test Case 8 — Throughput Performance with Two VMs in Series (OVS with DPDK-netdev, L3 Forwarding)

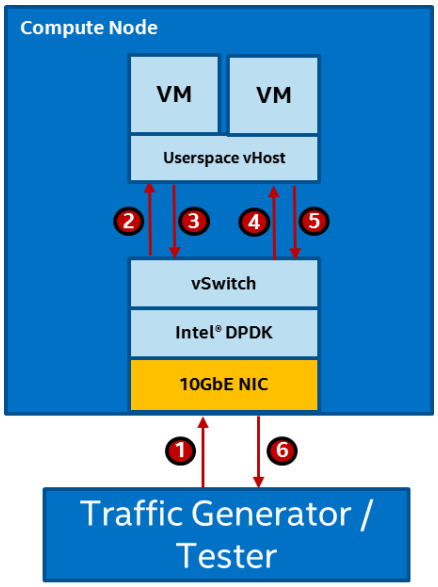

Figure 5–13. VM Throughput Performance with vSwitch — Two VMs in Series

Configuration (Done Manually):

1. Set up two VMs and connect to the vSwitch.
2. IP addresses of VMs are configured.
3. Flows get programmed to the vSwitch.

Data Path (Numbers Matching Red Circles):

1. The packet generator creates flows based on RFC 2544.
2. The vSwitch forwards the flows to the first vPort of VM1.
3. VM1 receives the flows and forwards them to VM2 through its second vPort (via the vSwitch).
4. The vSwitch forwards the flows to the first vPort of VM2.
5. VM2 receives the flows and forwards them back to the vSwitch through its second vPort.
6. The vSwitch forwards the flows back to the packet generator.

Test Configuration Notes:

- Traffic is bidirectional.
- Packets are not modified.

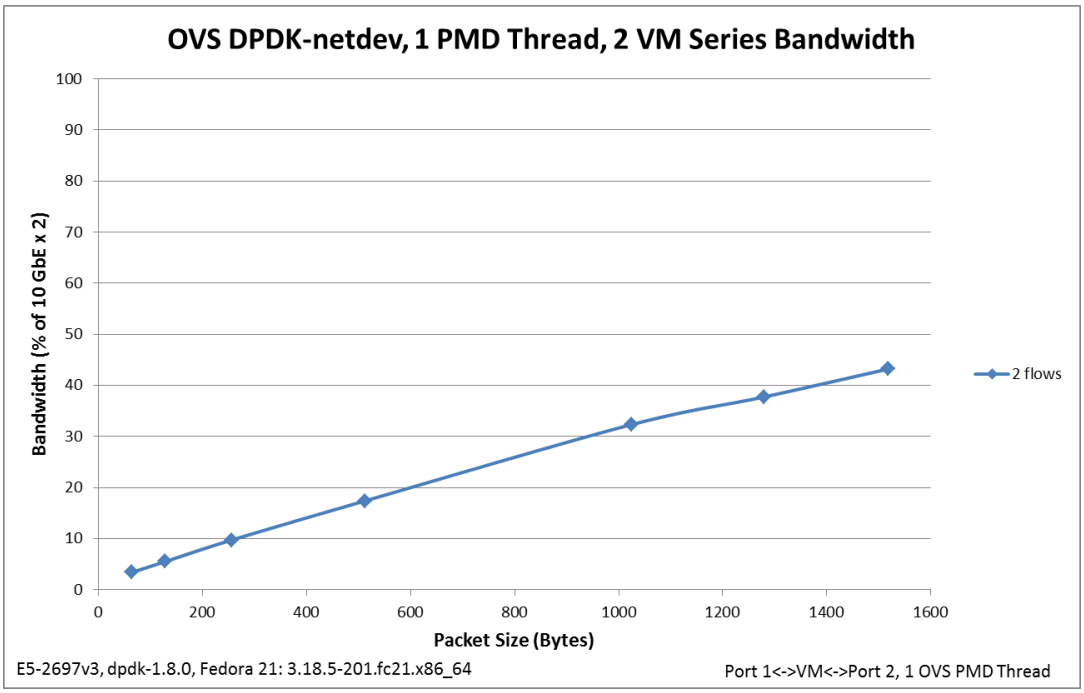

Figure 5–14. VM Throughput Performance — Two VMs in Series (OVS with DPDK-netdev)

5.4.3 Test Case 9 — Throughput Performance with Two VMs in Parallel (OVS with DPDK-netdev, L3 Forwarding)

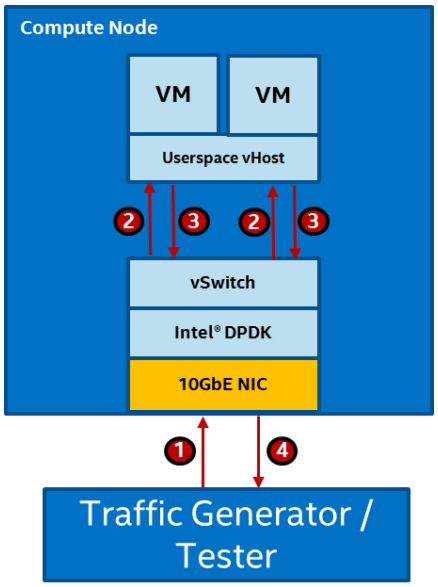

Figure 5–15. VM Throughput Performance with vSwitch — Two VMs in Parallel

Configuration (Done Manually):

1. Set up two VMs and connect to the vSwitch.
2. IP addresses of VMs are configured.
3. Flows get programmed to the vSwitch.

Data Path (Numbers Matching Red Circles):

1. The packet generator creates flows based on RFC 2544.
2. The vSwitch forwards the flows to the first vPort of VM1 and first vPort of VM2.
3. VM1 receives the flows and forwards out through its second vPort. VM2 receives the flows and forwards them out through its second vPort.
4. The vSwitch forwards the flows back to the packet generator.

Test Configuration Notes:

- Traffic is bidirectional.
- Packets not modified.

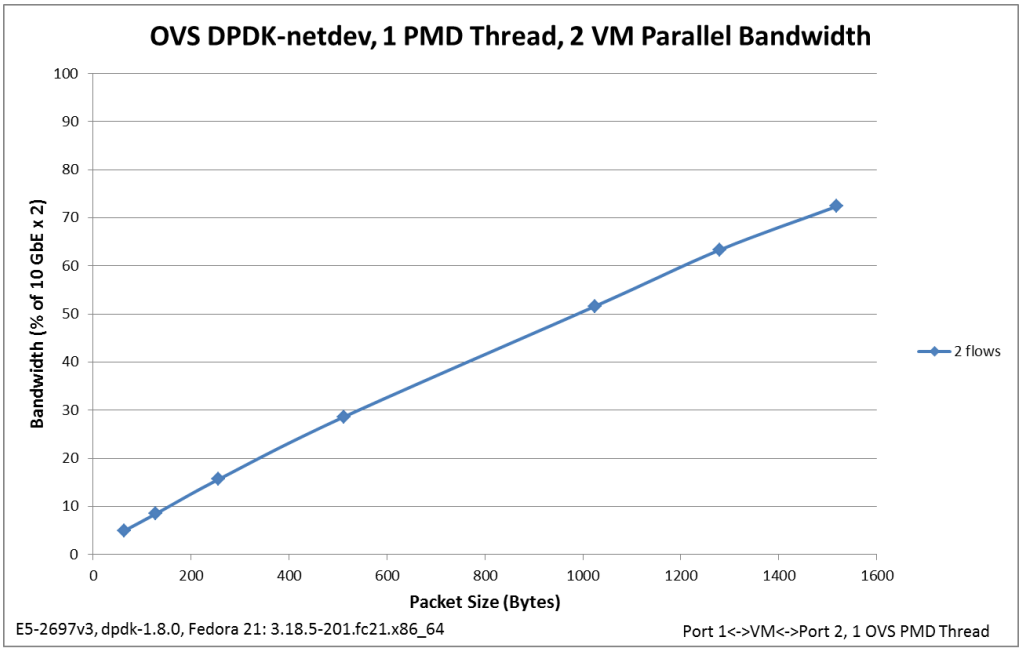

Figure 5–16. VM Throughput Performance — Two VMs in Parallel (OVS with DPDK-netdev)

Appendix A Host System Setup

Configuration of the host operating system affects performance of packet processing applications such as Open vSwitch using DPDK. This section contains information on how the installation and configuration were done to achieve the performance figures in this document.

A.1 Basic Fedora 21 Installation

A.1.1 Software Selection

Using the Fedora 21 Server DVD disk, apply the following options:

- Virtualization
- C Development Tools and Libraries
- Container Management
- Development Tools
- Headless Management
- RPM Development Tools
- System Tools

A.1.2 Disk Partition

A test system should just use a standard disk partition—ideally 250 GB or larger for ext4 / and then a swap partition. This allows the disk to be mounted and accessed (if needed) after it is no longer being used for testing.

A.1.3 Setting up the User during Install

1. Set the root password.
2. Create an additional administrative user.

A.2 Fedora 21 System Configuration

A.2.1 Disabling SELinux

Edit the SELinux configuration file and set to disabled:

```
# vi /etc/selinux/config
. . .
SELINUX=disabled
. . .
```

A.2.2 Speeding up the SSH Login

The **ssh daemon** configuration needs to be changed to help speed up the ssh login process. The VM login is faster than the default host, but this makes an almost instantaneous login prompt. The **ssh daemon** configuration is in */etc/ssh/sshd_config*:

Change the following setting:

```
# vi /etc/ssh/sshd_config
. . .
GSSAPIAuthentication yes
 . . .
```

To the following:

```
 . . .
 GSSAPIAuthentication no
 . . .
```

The login speed will change after the next reboot and on the **sshd** restart.

A.2.3 Adding Users

Add the new user and set the password:

```
# useradd -m user
# passwd user
```

For sudo privileges, add to the **wheel** group:

```
# vim /etc/group
. . .
wheel:x:10:mike,user
. . .
```

A.2.4 Configuring the Networking

A.2.4.1 Disabling Fedora's NetworkManager

Disable Fedora's NetworkManager and enable "normal" networking.

NetworkManager is good for desktops where network connections can change (e.g., LAN vs. WLAN), but on a server you usually do not change network connections:

```
# systemctl disable NetworkManager.service
# chkconfig network on
# systemctl restart network.service
# systemctl stop NetworkManager.service
```

A.2.4.2 Configuring the Ethernet Interfaces

The Ethernet interfaces are configured using configuration files in the */etc/sysconfig/network-scripts* directory. The names of the Ethernet devices are not the default eth0, eth1, eth2, ... names. These names need to be set in the configuration file using the DEVICE parameter. When setting a new device, the original configuration file from the default install needs to be removed from the */etc/sysconfig/network-scripts* directory. The devices names are matched to the network device created with the VM by having the HWADDR set with the Ethernet address.

The following is a sample */etc/sysconfig/network-scripts/ifcfg-eth0* file used for the first management interface:

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE="Ethernet"
BOOTPROTO="static"
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
NAME="eth0"
ONBOOT="yes"
HWADDR="00:1E:67:23:CD:73"
IPADDR="10.4.0.180"
NETMASK="255.255.248.0"
GATEWAY="10.4.0.1"
DNS1="8.8.8.8"
DNS2="4.2.2.2"
```

The DNS and gateway values are set to `eth0` in this example. The system should be rebooted to make sure the Ethernet devices come up with the correct configuration. There are methods to dynamically reinitialize the network. Testing here resulted in reinitializing network settings, but failed to initialize correctly on a system reboot. The system reboot should <u>not</u> be skipped.

A.2.5 Configuring the Network Management Bridge

A Linux network bridge can be used for creating a management network bridge to the VMs. This is the preferred network setup for VM network performance testing.

```
# cat ifcfg-br-mgt
TYPE=Bridge
DEVICE=br-mgt
NM_CONTROLLED=no
IPADDR=10.4.0.180
NETMASK=255.255.248.0
GATEWAY=10.4.0.1
DNS1=8.8.8.8
DNS2=4.2.2.2
ONBOOT=yes
STP=no
```

```
# cat ifcfg-eth0
DEVICE=eth0
HWADDR=00:1e:67:4f:d2:82
NM_CONTROLLED=no
BRIDGE=br-mgt
ONBOOT=yes
```

In the example above, the DNS and gateway values are moved from `eth0` to the bridge. A system reboot should be used to verify the new network settings and should <u>not</u> be skipped.

A.2.6 Setting and/or Checking the Host Name

The system host name is put in */etc/hostname*. Example name:

```
# vi /etc/hostname
# cat /etc/hostname
Fed21.test
```

A.3 System Server and Services Configuration

A.3.1 Disabling the Firewall

Disable the firewalld service:

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

A.3.2 Disabling irqbalance

The irqbalance daemon moves interrupt processing across the cores. The daemon works well for a general system and is enabled in a default install. Without irqbalance enabled or specific interrupt affinitization, most interrupt processing generally executes on CPU core 0.

During the system runtime, it can be terminated using the following command:

```
# killall irqbalance
```

The interrupt balancer, however, can disrupt certain types of performance-oriented software, as in the case of a DPDK program, by moving interrupt processing to the CPU cores, which are reserved for running the real-time software. To prevent the movement of the interrupt processing across the cores during runtime, the irqbalance should be disabled on system startup. The kernel boot-line argument "`isolcpus=1...n`" may also prevent the interrupt request (IRQ) from running on listed cores, however, this was <u>not</u> verified. To disable irqbalance on startup:

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

After issuing the command, reboot the system to restore the base IRQ to the CPU core mapping.

A.4 Installing the Baseline Package

A.4.1 Installing Order to Get Tools in Correctly

Disabling the kernel update at the beginning prevents all of the target tools from being installed correctly.

The system should be created in the following order:

1. Install new system
2. Basic configuration
3. Install development tools
4. Install target kernel
5. Disable kernel update (by yum)
6. Update system
5. Cleanup system

A.4.2 Installing VIM

```
# yum -y install vim
```

A.4.2 Installing tunctl

The VMs need tunnels into the bridge; `tunctl` provides support:

```
# yum -y install tunctl
```

A.4.4 Install Development Tools

```
# yum -y install rpmdevtools yum-utils ncurses-devel gcc \
make qt3-devel libXi-devel gcc-c++ openssl-devel coreutils \
kernel-devel
```

A.4.5 DPDK Build Tools

Recommended packages in addition to the development tools for the DPDK build:

```
# yum -y install glibc.i686 libgcc.i686 libstdc++.i686 fuse \
fuse-devel glibc-devel.i686 kernel-devel libcap-devel
```

A.4.7 Installing Screen

The screen program allows a persistent text screen to be created that can be detached and later reattached without closing the session. (This is very useful for testing.)

```
# yum -y install screen
```

`Terse Screen` command summary:

- `screen -S <name>` // Start new screen session called: <name>
- `screen -h` // List command summary
- `screen -ls` // List current screen sessions running
- `screen -D -R <name>` // Forcibly detach remote session attach
- `screen -x <name>` // Attach to a session already attached (multiplexed)
- `<control>-a d` // Detach from screen task

A.4.8 Installing Qemu-KVM

```
# yum -y install virt-manager virt-viewer virt-install libvirt \
  libvirt-python python-virtinst bridge-utils kvm qemu-kvm
```

Reboot and then check:

```
# lsmod | grep kvm
```

```
 kvm_intel             142999  0
 kvm                   444322  1 kvm_intel
 # systemctl start libvirtd.service
# systemctl enable libvirtd.service
# virsh -c qemu:///system list
  Id    Name                           State
 ----------------------------------------------------
```

A.4.9 CPU Sensors and Tools (Optional)

```
# yum -y install lm_sensors kernel-tools
```

Check your CPUs:

```
# numactl --hardware
# cpupower -c all monitor | head
# sensors | head
```

A.4.10 Disabling Kernel Update

To prevent the kernel from being updated when installing components, an exclusion line can be added to */etc/yum.conf.* This is useful when targeting a particular kernel that would be replaced by the newer kernel during an update.

```
# vi /etc/yum.conf
```

Add:

```
exclude=kernel*
```

Display the current kernel version running:

```
# echo `uname -r`
```

This also prevents installing a kernel through rpm local installs as well as any kernel package installs.

The default Fedora 21 install kernel 3.17.4 remains if the kernel update is excluded. If the update is not excluded, the update moves to the latest kernel distribution. At the time of this release, the kernel updated to 3.18.7 when the update was not excluded.

A.4.11 Updating the Operating System

Update the system and packages:

```
yum update -y
```

This also updates the kernel. It will exclude the kernel update, however, if the kernel update is disabled as described in the previous section.

A.4.12 Installing Cleanup

This removes extra data after an install or update. Apply as needed:

```
# yum clean all
```

A.5 Configuring the Kernel Boot Parameters

A.5.1 Hugepage Configuration

Multiple 1 GB and 2 MB hugepage sizes are supported, but the 1 GB hugepage support can only be enabled in the Linux boot line. The following kernel boot line enables and allocates both 1 GB and 2 MB hugepage memory:

```
default_hugepagesz=1GB hugepagesz=1GB hugepages=16 hugepagesz=2M hugepages=2048
```

In this case, the default hugepage size is 1 GB; 16 GB of 1 GB hugepage memory are allocated and 4 GB of 2 MB hugepages are allocated. The boot-line hugepage memory allocation is divided across Non-Uniform Memory Access (NUMA) nodes. For a two NUMA node system, half the memory is allocated from each NUMA node. For this example, on two sockets, eight 1 GB hugepages are allocated on NUMA node 0 and another 8 GB on NUMA node 1; 1024 2 MB pages are allocated on NUMA node 0 and another 1024 2 MB pages on NUMA node 1.

When multiple physical processors are inserted into the system, the memory attached to each processor forms a memory NUMA node. When using the general hugepage controls, the kernel attempts to distribute the hugepage pool over all the allowed NUMA nodes. If insufficient memory is on a NUMA node, it is skipped silently and the skipped memory page count is added to counts to be allocated on other NUMA nodes.

A.5.1.1 Dynamically Allocating 2 MB Hugepage Memory for NUMA Nodes

The 2 MB hugepages are set up on a per memory NUMA node with a page count set for each NUMA node. The following displays the number of memory pages for a system with two NUMA nodes:

```
# cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
1024
# cat /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
1024
```

There are currently 2048 2 MB pages in the memory pool with half for each NUMA node. For example, changing to 2048 2 MB pages for NUMA node 0 can be done by writing the count to the `nr_hugepages` variable for NUMA node 0:

```
# echo 2048 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
```

Reviewing the NUMA node information:

```
# cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
2048
# cat /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
1024
```

Notes:

1. Hugepage memory may not be added, if insufficient memory is available. Adding huge memory should be done before the system becomes heavily loaded with processes and/or VMs.

2. Unless insufficient NUMA memory is available and memory of another NUMA node is available, the Linux system allocates memory from the NUMA node that the task is running on. Controlling the NUMA node the task is running on and having sufficient NUMA node memory allows control of which NUMA node hugepage memory is allocated on.

A.5.2 Isolating the CPU Core

The CPU virtual processor cores can be removed from the Linux task scheduler so that some of these virtual processor cores can be reserved for real-time processing tasks. Real-time processing does not work as expected, if all CPU cores are assigned core-level, real-time tasks. Because the default Linux interrupt core is 0, it is recommended to always leave CPU 0 for general Linux system processing. If this is not done, all the interrupt processing needs to be moved to the non-real-time core before starting the real-time task.

If hyperthreading is enabled, both hyperthreaded cores pairs should be in the isolated core list. Full non-hyperthreaded core performance can be achieved by leaving the associated hyperthreaded core idle.

The following boot command-line argument isolates CPU physical cores 1 to 12 with associated hyperthreaded cores for the E5-2697 (14 core CPU) installed. Cores 0 and 13 with associated hyperthreaded cores are available for the Linux host system. The full second physical CPU socket cores are left for Linux system use in this case.

```
isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,28,29,30,31,32,33,34,35,36,37,38,39
```

The Linux command `taskset` can be used during program startup or for running programs to move a task or thread to the target CPU core.

For the E5-2699v3 (18-core processor) with hyperthreading enabled, the isolcpus may look more like this:

```
isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,28,29,37,38,39,40,41,42,43,44,45,46
```

A.5.3 Configuring the IOMMU

The IOMMU is needed for PCI passthrough and SR-IOV to pass a hardware PCI device to a VM. Enabling the IOMMU adds extra system overhead and security for all physical I/O devices in the system. When the IOMMU is not needed, it should be disabled by removing IOMMU arguments from the Linux boot command line and then rebooted. To enable IOMMU, add the following arguments to the Linux boot command line:

```
iommu=pt intel_iommu=on
```

Without `iommu=pt`, the host I/O operates much slower than when it is present.

A.5.4 Editing the Default Grub Configuration

The hugepage configuration can be added to the default configuration file */etc/default/grub* by adding to the `GRUB_CMDLINE_LINUX` and the grub configuration file regenerated to get an updated configuration file for Linux boot:

```
# vim /etc/default/grub            // edit file

. . .
GRUB_CMDLINE_LINUX="default_hugepagesz=1GB hugepagesz=1GB hugepages=16 \
hugepagesize=2m hugepages=2048 isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,28,29,30,31,\
32,33,34,35,36,37,38,39 iommu=pt intel_iommu=on ..."
. . .
```

After editing, the new grub.cfg boot file needs to be regenerated:

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

A.5.5 Verifying Kernel Boot Configuration

The system needs to be rebooted for the kernel changes to take effect. After reboot, the kernel command line can be checked looking at kernel messages with `dmesg`:

```
# dmesg |grep command
[    0.000000] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-3.18.5-201.fc21.x86_64
root=UUID=cc17be46-707c-414a-8c16-11ecda5905be ro default_hugepagesz=1GB
hugepagesz=1GB hugepages=16 hugepagesz=2M hugepages=2048
isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,28,29,30,31,32,33,34,35,36,37,38,39 rhgb quiet
```

The `dmesg` only outputs the last system messages that are present in a limited reserved memory for system messages. If the kernel boot command line is not present, then search the full log file */var/log/messages*.

The memory hugepage information also can be checked:

```
# cat /proc/meminfo
. . .
HugePages_Total:      16
HugePages_Free:       16
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:    1048576 kB
. . .
```

Only the hugepage count for the default memory size is shown. For the case of having both 1 GB and 2 MB hugepages, only the 1 GB hugepage count is shown with */proc/meminfo*.

A.6 Configuring System Variables (Host System Configuration)

```
# echo "# Disable Address Space Layout Randomization (ASLR)" > /etc/sysctl.d/
aslr.conf
# echo "kernel.randomize_va_space=0" >> /etc/sysctl.d/aslr.conf
# echo "# Enable IPv4 Forwarding" > /etc/sysctl.d/ip_forward.conf
# echo "net.ipv4.ip_forward=1" >> /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
```

Appendix B VM System Setup

What is in the VM and how it is configured can affect how well the VM performs. This is the install and setup used for some tests in this document.

There are many variations to creating a VM. Here is one method for installing from the Fedora 21 server DVD disk image.

B.1 Creating a qcow2 Virtual Disk

A VM disk file of 20 GB is made using the `qemu-img` command:

```
# qemu-img create -f qcow2 Fed21-test.qcow2 20G
```

There are many parameters as well as many variations of startup scripts. In this case, the ISO install disk is on the hard disk as a file *(/root/Fedora-Server-DVD-x86_64-21.iso*) and passed to the VM as a **cdrom** device.

The host has one bridge, **br-mgt**, in this script. Assuming the **br-mgt** is bridging the primary control network and assigning a non-conflicting address for the VM interface on **br-mgt**, the VM should be able to connect to the Internet, if the host interface of the bridged management interface can connect. This is useful for updating packages on the VM from the Internet. The VM startup script is followed by the network exit script for the **br-mgt** network bridge:

```
# cat Fed21_VM_Install.sh
#!/bin/sh

vm=/vm/Fed21-mp.qcow2
vm_name=Fed21_Test_VM
vnc=14
n1=tap55
bra=br-mgt
dn_scrp_a=/vm/vm_ctl/br-mgt-ifdown
mac1=00:00:14:42:04:29
cd=/root/Fedora-Server-DVD-x86_64-21.iso

if [ ! -f $vm ];
then
    echo "VM $vm not found!"
else
    echo "VM $vm started! VNC: $vnc, net0: $n1, net1: $n2"
    tunctl -t $n1
    brctl addif $bra $n1
    ifconfig $n1 0.0.0.0 up
    /usr/bin/qemu-system-x86_64 -enable-kvm -m 4096 -smp 4 -cpu host \
-net nic,model=virtio,netdev=eth0,macaddr=$mac1 \
-netdev tap,ifname=$n1,id=eth0,vhost=on,script=no,downscript=$dn_scrp_a \
-drive file=$vm,if=virtio -cdrom $cd -vnc :$vnc -name $vm_name &
fi
```

```
# cat br-mgt-ifdown
#!/bin/sh

bridge='br-mgt'
/sbin/ifconfig $1 0.0.0.0 down
brctl delif  ${bridge} $1
```

The QEMU executable is named differently on different Linux systems and/or system versions. On a Fedora20 and Ubuntu 12.04 system, the QEMU executable is */usr/bin/qemu-system-x86_64*. On a CentOS 6.3 system, the QEMU executable is */usr/libexec/qemu-kvm.*

After the install, the `-cdrom $cd` can be removed to make a run script because the install ISO disk is no longer needed.

B.2 Selecting VM Install Software

From the Fedora 21 Server install disk, the VM can be created by selecting the following packages:

- Guest Agents
- C Development tools and Libraries
- Development Tools

The VM that is created with this install method boots moderately fast.

B.2.1 Standard Disk Partition with Auto Partitioning

Auto partitioning for a standard disk creates **/boot, /,** and **swap** partitions:

- Install in standard disk partition (auto partitioning)
- Non-encrypted disk

B.2.2 VM Disk Format to Use with OpenStack

A VM for an OpenStack install should not be LVM, or have a **/boot** partition or swap partition. The VM disk should be qcow2 and have only one partition with a **/** mount point. The disk partition needs to be manually created.

B.2.3 While Installing

Set the root password and create another user with administrator privileges.

B.3 Fedora 21 System Configuration

B.3.1 Disabling SELinux

Edit the SELinux configuration file and set to disabled:

```
# vi /etc/selinux/config

. . .
SELINUX=disabled
. . .
```

B.3.2 Speeding up the SSH Login

The **ssh daemon** configuration needs to be changed to help speed up the ssh login process. The VM login is faster than the default host, making an almost instantaneous login prompt. The **ssh daemon** configuration is in */etc/ssh/sshd_config*.

Change the following setting:

```
# vi /etc/ssh/sshd_config

 . . .
 GSSAPIAuthentication yes
 . . .
```

To the following:

```
 . . .
 GSSAPIAuthentication no
 . . .
```

The login speed changes after next reboot, but also changes on the **sshd** restart.

B.3.3 Adding Users

Add a new user and set the password:

```
# useradd -m user
# passwd user
```

For sudo privileges, add to the **wheel** group:

```
# vim /etc/group
. . .
wheel:x:10:mike,user
. . .
```

B.3.4 Configuring Networking

B.3.4.1 Configuring Ethernet Interfaces

Ethernet interfaces are configured using the configuration files in the */etc/sysconfig/network-scripts* directory. The names of the Ethernet devices are not the default **eth0**, **eth1**, **eth2**, ... names. These names need to be set in the configuration file, with configuration file with by the same name using the **DEVICE** parameter. When setting a new device, the original configuration file from the default install, needs to be removed from the */etc/sysconfig/network-scripts* directory. The devices names are matched to the network device created with the VM by having the **HWADDR** set with the Ethernet address.

The following is sample */etc/sysconfig/network-scripts/ifcfg-eth0* file used for the first management virtio interface in a VM:

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE="Ethernet"
BOOTPROTO="static"
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
NAME="eth0"
UUID="c1548663-a338-484f-a08b-63ef6bbc7548"
ONBOOT="yes"
HWADDR="00:00:14:42:04:29"
IPADDR="10.4.5.180"
NETMASK="255.255.248.0"
GATEWAY="10.4.0.1"
DNS1="8.8.8.8"
DNS2="4.2.2.2"
```

B.3.4.2 Setting and/or Checking the Host Name

The system host name is put in */etc/hostname.* Example name:

```
# vi /etc/hostname
# cat /etc/hostname
Fed21vm.test
```

B.3.5 Checking the NFS Client Mount

A check of a mount to a Network File System (NFS) server on the same network can be done (assuming 10.4.0.251:/storage on the current network) with the following command:

```
# mkdir /xusr
# mount -t nfs 10.4.0.251:/storage /xusr
```

B.4 Configuring the System Server and Services

B.4.1 Disabling the Firewall

To disable the `firewalld` service, issue the following command:

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

B.4.2 Disabling irqbalance

The irqbalance daemon moves interrupt processing across the cores and works well for a general system and and is enabled in a default install. Without the irqbalance enabled or specific interrupt affinitization most interrupt processing generally executes on CPU core 0. For instructions, refer to section A.3.2, Disabling irqbalance.

B.5 Installing the Baseline Package

B.5.1 Ordering the Installation to Correctly Install Tools

Disabling the kernel update at the beginning prevents all of the target tools from being installed correctly. The following system creation order should be used:

- Install new system
- Basic configuration
- Install development tools
- Install target kernel
- Disable kernel update (by yum)
- Update system
- Cleanup system

B.5.2 Installing VIM

```
# yum -y install vim
```

B.5.3 Installing Development Tools

```
# yum -y install rpmdevtools yum-utils ncurses-devel gcc \
make qt3-devel libXi-devel gcc-c++ openssl-devel coreutils \
kernel-devel
```

B.5.4 DPDK Build Tools

Recommended packages in addition to the development tools for DPDK build:

```
# yum -y install glibc.i686 libgcc.i686 libstdc++.i686 fuse \
fuse-devel glibc-devel.i686 kernel-devel libcap-devel
```

B.5.5 Installing Screen

The screen program allows a persistent text screen to be created that can be detached and then reattached without closing the session. This is very useful for testing. For instructions, refer to section A.4.7, Installing Screen.

B.5.6 Disabling Kernel Update

To prevent the kernel from being updated when installing components, an exclusion line can be added to */etc/yum.conf.* This is useful when targeting a particular kernel to be replaced by a newer kernel during an update.

```
# vi /etc/yum.conf
```

Add:

```
exclude=kernel*
```

Display current kernel running:

```
# echo `uname -r`
```

This also prevents installing a kernel through rpm local installs as well as any kernel package installs.

If the kernel update is excluded, the default Fedora 21 install kernel 3.17.4 remains. If the update is not excluded, the update moves to the latest kernel distribution. At the time of writing this, the kernel was updated to 3.18.7 when the kernel update was not excluded.

B.5.7 Updating the Operating System

Update the system and packages:

```
yum update -y
```

This also updates the kernel. If the update is disabled as described in the previous section, however, it excludes the kernel update.

B.5.7 Installing Cleanup

This removes extra data after an install or update:

```
# yum clean all
```

Apply as needed.

B.6. Configuring Kernel Boot Parameters

B.6.1 Configuring the Hugepage

Although multiple 1 GB and 2 MB hugepage sizes are supported, only 1 GB hugepage support can be enabled in the Linux boot line. The following kernel boot line enables and allocates both 1 GB and 2 MB hugepage memory:

```
default_hugepagesz=1GB hugepagesz=1GB hugepages=1 hugepagesz=2M hugepages=256
```

In this case, the default hugepage size is 1 GB and 1 GB of 1 GB hugepage memory is allocated and 0.5 GB of 2 MB hugepages is allocated. It requires a VM created from a minimum of 3 GB of 1 GB hugepage memory to have one valid 1 GB of hugepage memory in the VM. If a VM is not created with the largest hugepages needed by the VM, the VM's hugepage memory is generated from smaller memory pages. This results in operating with a smaller page size that generally has lower performance for applications designed to use larger hugepages.

B.6.2 Isolating the Virtual CPU Core

The CPU virtual processor cores can be removed from the VM's Linux task scheduler so that some of these virtual processor cores can be reserved for real-time processing tasks. For example, vCPU cores 1 and 2 can be used in a DPDK application, while vCPU core 0 is left available for VM Linux operations and VM kernel interrupt processing. Real-time processing does not work as expected, if all vCPU processors are assigned core-level real-time tasks. Because the default Linux interrupt core is 0, it is recommended to always use vCPU 0 for general Linux processing. If cVPU is not used, all the interrupt processing needs to be moved to the non-real-time virtual core before starting the real-time task.

The following command is for a 3 vCPU VM, leaving vCPU-0 for the VM Linux and vCPU-1 and vCPU-2 for running real-time tasks:

```
isolcpus=1,2
```

The Linux `taskset` command can be used during program startup or for running programs to move a task or thread to the target vCPU.

B.6.3 Editing the Default Grub Configuration

The hugepage configuration can be added to the default configuration file */etc/default/grub* by adding to the `GRUB_CMDLINE_LINUX` and the grub configuration file regenerated to get an updated configuration file for the Linux boot:

```
# vim /etc/default/grub            // edit file

. . .
GRUB_CMDLINE_LINUX="default_hugepagesz=1GB hugepagesz=1GB hugepages=1 \
hugepagesize=2m hugepages=512 isolcpus=1,2 ..."
. . .
```

After editing, the new *grub.cfg* boot file needs to be regenerated:

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

The system needs to be rebooted for the kernel changes to take effect. After reboot, the kernel command line can be checked:

```
# dmesg |grep command
[    0.000000] Kernel command line: BOOT_IMAGE=/vmlinuz-3.18.7-200.fc21.x86_64
root=/dev/mapper/fedora--server-root ro default_hugepagesz=1GB hugepagesz=1GB
hugepages=1 hugepagesz=2M hugepages=512 isolcpus=1,2 rd.lvm.lv=fedora-server/root
rd.lvm.lv=fedora-server/swap rhgb quiet
```

B.7 Preparing to Use OpenStack

OpenStack Linux image requirements are per the image-guide:

- Disk partitions and resize root partition on boot (cloud-init)
- No hard-coded MAC address information
- SSH server running
- Disable firewall
- Access instance using ssh public key (cloud-init)
- Process user data and other metadata (cloud-init)

You can fulfill the requirement for some of these by installing the cloud-init package.

B.7.1 Installing Packages for the Cloud Operation

Issue the following command to install packages for the cloud operation:

```
# yum install cloud-utils cloud-init
```

B.7.2 Setting up the Network for the Cloud Operation

Set Ethernet eth0 configuration */etc/sysconfig/network-scripts/ifcfg-eth0* to **dhcp** initialization and remove hardware address HWADDR for the OpenStack startup. A sample configuration that allows OpenStack install for basic testing is as follows:

```
# cat ifcfg-eth0

TYPE="Ethernet"
BOOTPROTO="dhcp"
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
# IPV6INIT="yes"
# IPV6_AUTOCONF="yes"
# IPV6_DEFROUTE="yes"
# IPV6_PEERDNS="yes"
# IPV6_PEERROUTES="yes"
# IPV6_FAILURE_FATAL="no"
NAME="eth0"
UUID="10647d04-2cb8-44d3-930d-833d50381c13"
```

```
ONBOOT="yes"
# HWADDR="00:00:14:42:04:29"
# PEERDNS="yes"
# PEERROUTES="yes"
```

This disables IPV6 and is only shown as an example and may not be the ideal settings for OpenStack. The universally unique identifier (UUID) that was created during the install process should be kept unless duplicating VMs.

B.7.3 Changing the Koji Fedora Kernel Version

To have the system use a certain target Fedora kernel version that is different from the current kernel version, a koji build kernel can be used. The documentation in this report is for tests conducted using Fedora 20, but the same method with a different version can be used for Fedora 21.

If the new target kernel is not to be updated during the package install, then the kernel component update needs to be disabled with the line `exclude=kernel*` in */etc/yum.conf*.

The example below targets a specific koji kernel build, kernel-3.14.8-200.fc20.

For the test installation:

1. Go to the koji repository: http://koji.fedoraproject.org/koji/packageinfo?packageID=8.
2. Select the target kernel (kernel-3.14.8-200.fc20) from the Fedora koji builds. Clicking the kernel link, opens the following page:

   http://koji.fedoraproject.org/koji/buildinfo?buildID=538530
3. From the 64 bit x86 build section x86_64, download the following files from the build:

   ```
   kernel-3.14.8-200.fc20.x86_64.rpm
   kernel-devel-3.14.8-200.fc20.x86_64.rpm
   kernel-headers-3.14.8-200.fc20.x86_64.rpm
   kernel-modules-extra-3.14.8-200.fc20.x86_64.rpm
   kernel-tools-3.14.8-200.fc20.x86_64.rpm
   kernel-tools-libs-3.14.8-200.fc20.x86_64.rpm
   ```
4. Save the files in the NFS drive, e.g., */xusr/Fedora20/koji-kernel-3.14.8.*

B.7.4 Updating the Kernel from the Koji Build

1. Verify the current kernel is before the target kernel:

   ```
   # echo `uname -r`
   3.11.10-301.fc20.x86_64
   ```
2. Copy target kernel update to the local machine (assume NFS mount and path):

   ```
   # dir /xusr/Fedora20/koji-kernel-3.14.8
   kernel-headers-3.14.8-200.fc20.x86_64.rpm
   kernel-3.14.8-200.fc20.x86_64.rpm
   kernel-modules-extra-3.14.8-200.fc20.x86_64.rpm
   kernel-devel-3.14.8-200.fc20.x86_64.rpm
   kernel-tools-3.14.8-200.fc20.x86_64.rpm
   kernel-tools-libs-3.14.8-200.fc20.x86_64.rpm
   ```

```
# cd ~
# cp /xusr/Fedora20/koji-kernel-3.14.8/* .
```

3. Review kernel components that are installed to determine which of these kernel modules need to be installed:

```
# yum list installed|grep kernel
abrt-addon-kerneloops.x86_64          2.2.1-2.fc20              @updates
kernel.x86_64                         3.11.10-301.fc20          @anaconda
kernel.x86_64                         3.14.9-200.fc20           @updates
kernel-devel.x86_64                   3.11.10-301.fc20          @anaconda
kernel-devel.x86_64                   3.14.9-200.fc20           @updates
kernel-headers.x86_64                 3.14.9-200.fc20           @updates
kernel-modules-extra.x86_64           3.14.9-200.fc20           @updates
kernel-tools.x86_64                   3.14.9-200.fc20           @updates
kernel-tools-libs.x86_64              3.14.9-200.fc20           @updates
libreport-plugin-kerneloops.x86_64    2.2.2-2.fc20              @updates
```

   In this case all six kernel components for kernel version 3.14.9 are installed. Because the target kernel is for an earlier build 3.14.8, all of these components need to be removed and the same components of the target kernel need to be installed per section B.7.5, Moving to an Earlier Kernel Version Using the Koji Build.

4. Enable the update of the kernel:

```
# vi /etc/yum.conf
```

5. Change from:

```
exclude=kernel*
```

   To (comment out the kernel exclude line to enable the kernel update):

```
#exclude=kernel*
```

6. Install the kernel components:

```
# yum localinstall kernel-3.14.8-200.fc20.x86_64.rpm
# yum localinstall kernel-devel-3.14.8-200.fc20.x86_64.rpm
# yum localinstall kernel-headers-3.14.8-200.fc20.x86_64.rpm
# yum localinstall kernel-modules-extra-3.14.8-200.fc20.x86_64.rpm
# yum localinstall kernel-tools-3.14.8-200.fc20.x86_64.rpm
# yum localinstall kernel-tools-libs-3.14.8-200.fc20.x86_64.rpm
```

7. Reboot and verify the new 3.14.8-200.fc20.x86_64 kernel is running:

```
# reboot
. . .
# echo `uname -r`
3.14.8-200.fc20.x86_64
```

   This results in the kernel library modules being installed:

```
# ls /lib/modules/
3.11.10-301.fc20.x86_64  3.14.8-200.fc20.x86_64
```

   The kernel includes files installed:

```
# ls /usr/src/kernels/
3.11.10-301.fc20.x86_64  3.14.8-200.fc20.x86_64
```

   And files for 3.14.8-200.fc20.x86_64 in the boot directory:

```
# ls -l /boot
. . .
-rw-r--r--  1 root root   141773 Jun 16 15:08 config-3.14.8-200.fc20.x86_64
-rw-------  1 root root  9660063 Jul 16 14:47 initramfs-3.14.8-
200.fc20.x86_64.img
-rw-------  1 root root  2854783 Jun 16 15:08 System.map-3.14.8-200.fc20.x86_64
-rwxr-xr-x  1 root root  5518328 Jun 16 15:08 vmlinuz-3.14.8-200.fc20.x86_64
```

This also updates the grub file for kernel booting `3.14.8-200.fc20.x86_64` being the default boot file.

Post-kernel install update

Do not reinstall kernel-related packages, because newer versions (e.g., kernel-devel) can get installed. If unsure of the kernel package you need, do a local install of the target kernel koji package. In some cases, it may be desirable to install all six koji kernel components even when they were not all installed for the earlier kernel.

1. Disable the update of the kernel to prevent being changed later:

   ```
   # vi /etc/yum.conf
   ```

2. Change from:

   ```
   # exclude=kernel*
   ```

   Back to:

   ```
   exclude=kernel*
   ```

3. Update the system for any new packages needed (kernel update needs to be disabled):

   ```
   # yum update -y
   # yum clean all
   ```

   Note: After the kernel update, a reinstall of some of the packages needs to be done so that all of the systems target packages install.

B.7.5 Moving to an Earlier Kernel Version Using the Koji Build

Note: It is advisable to use reinstall and upgrade and to avoid kernel downgrades, because this may make the system unstable. In the example below, a downgrade was successfully performed, however, this is not advised.

In this case, the current runtime kernel was already upgraded past the current target kernel:

```
# echo `uname -r`
3.15.4-200.fc20.x86_64
```

The system will not allow you install an older kernel then the current latest kernel.

```
# yum localinstall kernel-3.14.8-200.fc20.x86_64.rpm
. . .
Transaction check error:
  package kernel-3.15.4-200.fc20.x86_64 (which is newer than
kernel-3.14.8-200.fc20.x86_64) is already installed

Error Summary
-------------
```

```
################## Cannot install older kernel when newer one
```

1. To install an older kernel, the newer kernel needs to be removed. Since the system was installed from the install DVD, the original install has an earlier version. Do a reboot, and from the grub menu, boot the earlier kernel. After the boot, check that the earlier kernel is running:

   ```
   # echo `uname -r`
   3.11.10-301.fc20.x86_64
   ```

2. Determine what kernel modules need to be removed by determining what is installed:

   ```
   # yum list installed |grep kernel
   kernel.x86_64                    3.11.10-301.fc20              @anaconda
   kernel.x86_64                    3.15.4-200.fc20               @updates
   kernel-devel.x86_64              3.11.10-301.fc20              @anaconda
   kernel-devel.x86_64              3.15.4-200.fc20               @updates
   kernel-headers.x86_64            3.15.4-200.fc20               @updates
   ```

   In this case, three new kernel components are installed. The components installed are determined by the system packages that have been installed. If removing, using the general component's name, e.g., kernel-devel.x86_64, both 3.11.10 and 3.15.4 are removed. The two pieces of information must be combined to uninstall the specific version.

   For example: `kernel-devel.x86_64` and `3.15.4-200.fc20` are put together to get:

   ```
   kernel-devel-3.15.4-200.fc20.x86_64
   ```

3. Remove all newer kernel components:

   ```
   # yum remove kernel-3.15.4-200.fc20.x86_64
   # yum remove kernel-devel-3.15.4-200.fc20.x86_64
   # yum remove kernel-headers-3.15.4-200.fc20.x86_64.rpm
   ```

4. Install the same kernel components of the new target kernel per section B.7.4, Updating the Kernel from the Koji Build.

Appendix C Affinitization of Cores

C.1 Affinitization and Performance Tuning

To maximize network throughput, individual cores must be affinitized to particular tasks. This can be achieved by using either the `taskset` command on the host and/or by passing a core mask parameter to the VM application.

If the VM starts with three virtual CPUs (vCPU0, vCPU1, and vCPU2), the Linux operating system tasks must only use vCPU0. vCPU1 and vCPU2 are reserved for running the DPDK processes.

C1.1. Affinitization Using Core Mask Parameter in the qemu and test-pmd Startup Commands

The qemu, `test-pmd` startup commands offer a core mask parameter that can be set with a hex mask to ensure the tasks use specific cores.

Use core mask: -c 0x1 for `test-pmd` commands. This ensures that the DPDK task application in the VM uses vCPU1. Ensure by running the top command, if vCPU1 is used at 100%.

Even though the core mask is set to use two host cores for VMs vCPU0 and vCPU1, the `qemu` command allocates the vCPU0 and vCPU1 tasks on a single host core, primarily the first core of the specified core mask.

C1.2 Affinitizing Host Cores for VMs vCPU0 and vCPU1

1. Use the `taskset` command to pin specific processes to a core:

   ```
   # taskset -p <core_mask> <pid>
   ```

2. Ensure that the VM's vCPU0 and vCPU1 are assigned to two separate host cores.

   For example:

   - QEMU task for VM's vCPU0 uses core 4 (-c 0x30)
   - QEMU task for VM's vCPU1 uses core 5 (-c 0x30; taskset `-p 20 <pid_vcpu1>`)

Appendix D Building DPDK

D.1 Getting DPDK

The DPDK needs to be built differently for 1) the various test programs (e.g., DPDK L2 and L3 test programs), and 2) OVS with DPDK-netdev. The DPDK should be built in the same way for the host machine and VM. Appendix D.1.1 and Appendix D.1.2 provide information on two ways to obtain the DPDK source code.

D.1.1 Getting the DPDK Git Source Repository

The DPDK git repository was used to obtain the code for tests in this document because the latest available tar file changes often. Older tar files are more difficult to obtain compared to the target git code, which is obtained by just checking out with the correct commit id or tag:

```
# cd /usr/src
# git clone git://dpdk.org/dpdk
# cd /usr/src/dpdk
```

Need to check out the target DPDK version:

```
# git checkout -b test_v1.8.0 v1.8.0
```

The export directory for DPDK git repository is:

```
# export RTE_SDK=/usr/src/dpdk
```

D.1.2 Getting the DPDK Source Tar from DPDK.ORG

An alternative method to get the DPDK source is provided here as additional information.

Get DPDK release 1.8 and untar it:

1. Download DPDK 1.8 at: http://www.dpdk.org/download.
2. Move to the install location and the DPDK tar file to the destination, and then untar:

   ```
   # cd /usr/src
   # mv ~/dpdk-1.8.0.tar.gz .
   # tar -xf dpdk-1.8.0.tar.gz
   # cd /usr/src/dpdk-1.8.0
   ```

3. The specific DPDK version directory must be used for the code builds. The git repository DPDK directory is shown in the documentation below and needs to be changed for the DPDK 1.8.0 target directory:

   ```
   # export RTE_SDK=/usr/src/dpdk-1.8.0
   ```

D.2 Building DPDK for Applications

1. Move to the DPDK directory:

   ```
   # cd $RTE_SDK
   ```

2. Verify that no change have been made to the DPDK repository:

   ```
   # git diff -M -C
   #
   ```

3. If DPDK was built for a different configuration (i.e., OVS build), remove the old install:

   ```
   # make uninstall
   ```

4. Build DPDK:

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

   Note: The `ivshmem` option adds support for IVSHMEM and other interface options (user space vhost). Using this option keeps the build consistent across all test configurations.

D.3. Building DPDK for OVS

D.3.1 Building DPDK for OVS — Method A

1. Move to the DPDK directory:

   ```
   # cd $RTE_SDK
   ```

2. Set the `CONFIG_RTE_BUILD_COMBINE_LIBS=y` and `CONFIG_RTE_BUILD_COMBINE_LIBS=y` in the *config/common_linuxapp* file. It can be changed using a text editor or converting the git difference into a patch:

   ```
   # git diff -M -C
   diff --git a/config/common_linuxapp b/config/common_linuxapp
   index 2f9643b..cf211b8 100644
   --- a/config/common_linuxapp
   +++ b/config/common_linuxapp
   @@ -81,7 +81,7 @@ CONFIG_RTE_BUILD_SHARED_LIB=n
    #
    # Combine to one single library
    #
   -CONFIG_RTE_BUILD_COMBINE_LIBS=n
   +CONFIG_RTE_BUILD_COMBINE_LIBS=y
    CONFIG_RTE_LIBNAME="intel_dpdk"

    #
   @@ -372,7 +372,7 @@ CONFIG_RTE_KNI_VHOST_DEBUG_TX=n
    # fuse-devel is needed to run vhost.
    # fuse-devel enables user space char driver development
    #
   -CONFIG_RTE_LIBRTE_VHOST=n
   ```

```
+CONFIG_RTE_LIBRTE_VHOST=y
 CONFIG_RTE_LIBRTE_VHOST_DEBUG=n
```

3. Apply the patch by copying it into a file in the upper-level directory immediately above the DPDK directory (*/usr/src*). If the patch file is called *patch_dpdk_single_lib*, the following is an example of it being applied:

```
cd /usr/src/dpdk
patch -p1 < ../patch_dpdk_single_lib
```

4. If DPDK is built for a different configuration (i.e., DPDK Apps), remove old the install:

```
# make uninstall
```

5. Build DPDK:

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

D.3.2 Building DPDK for OVS — Method B

1. Move to the DPDK directory:

```
# cd $RTE_SDK
```

2. Verify that no change have been made to the DPDK repository:

```
# git diff -M -C
#
```

3. If DPDK was built for a different configuration (i.e., DPDK Apps), remove the old install:

```
# make uninstall
```

4. Build DPDK:

```
# make install T=x86_64-ivshmem-linuxapp-gcc CONFIG_RTE_LIBRTE_VHOST=y
CONFIG_RTE_BUILD_COMBINE_LIBS=y
```

Appendix E Host DPDK L2 and L3 Forwarding Applications

E.1 Building DPDK and L3/L2 Forwarding Applications

The DDPK is installed with IVSHMEM support. It can be also installed without IVSHMEM (see DPDK documentation in Appendix L, References).

1. If previously built for OVS, DPDK needs to be uninstalled first:

```
# make uninstall
# make install T=x86_64-ivshmem-linuxapp-gcc
```

2. Assuming the DPDK git repository, change the DPDK paths for the tar install:

```
# export RTE_SDK=/usr/src/dpdk
# export RTE_TARGET=x86_64-ivshmem-linuxapp-gcc
```

3. Build l3fwd:

   a. Change the NIC I/O queue sizes to 2048 either by editing or making the following into a patch:

```
# git diff -M -C
diff --git a/examples/l3fwd/main.c b/examples/l3fwd/main.c
index 918f2cb..d9d8ef4 100644
--- a/examples/l3fwd/main.c
+++ b/examples/l3fwd/main.c
@@ -156,8 +156,8 @@
 /*
  * Configurable number of RX/TX ring descriptors
  */
-#define RTE_TEST_RX_DESC_DEFAULT 128
-#define RTE_TEST_TX_DESC_DEFAULT 512
+#define RTE_TEST_RX_DESC_DEFAULT 2048
+#define RTE_TEST_TX_DESC_DEFAULT 2048
 static uint16_t nb_rxd = RTE_TEST_RX_DESC_DEFAULT;
 static uint16_t nb_txd = RTE_TEST_TX_DESC_DEFAULT;
```

   b. Build it:

```
# cd /usr/src/dpdk/examples/l3fwd/
# make
```

4. Build l2fwd:
   a. Change NIC I/O queue sizes to 2048 and increase the buffer count to 16384 either by editing or making the following into a patch and applying:

```
# git diff -M -C
diff --git a/examples/l2fwd/main.c b/examples/l2fwd/main.c
index e684234..577b235 100644
--- a/examples/l2fwd/main.c
+++ b/examples/l2fwd/main.c
@@ -73,7 +73,7 @@
 #define RTE_LOGTYPE_L2FWD RTE_LOGTYPE_USER1

 #define MBUF_SIZE (2048 + sizeof(struct rte_mbuf) + RTE_PKTMBUF_HEADROOM)
-#define NB_MBUF   8192
+#define NB_MBUF   16384

 #define MAX_PKT_BURST 32
 #define BURST_TX_DRAIN_US 100 /* TX drain every ~100us */
@@ -81,8 +81,8 @@
 /*
  * Configurable number of RX/TX ring descriptors
  */
-#define RTE_TEST_RX_DESC_DEFAULT 128
-#define RTE_TEST_TX_DESC_DEFAULT 512
+#define RTE_TEST_RX_DESC_DEFAULT 1024
+#define RTE_TEST_TX_DESC_DEFAULT 1024
 static uint16_t nb_rxd = RTE_TEST_RX_DESC_DEFAULT;
 static uint16_t nb_txd = RTE_TEST_TX_DESC_DEFAULT;
```

   b. Build it:

```
# cd /usr/src/dpdk/examples/l2fwd/
# make
```

E.2 Running the DPDK L3 Forwarding Application

The L3fwd program forwards the packet without modifying it:

1. Build DPDK and l3fwd as described in section Appendix D.2, Building DPDK for DPDK Applications.
2. Reboot the system, and check the kernel boot line for 1 GB hugepages, iosolcpu setting for target cores (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39 for a 14-core CPU with hyperthreading enabled) and IOMMU disabled:

```
# dmesg |grep command
[    0.000000] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-3.18.5-
201.fc21.x86_64 root=UUID=cc17be46-707c-414a-8c16-11ecda5905be ro
default_hugepagesz=1GB hugepagesz=1GB hugepages=16 hugepagesz=2M hugepages=2048
isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,28,29,30,31,32,33,34,35,36,37,38,39 rhgb
quiet
```

3. Initialize the host hugepage memory after reboot:

```
# mount -t hugetlbfs nodev /dev/hugepages
# mkdir /dev/hugepages_2mb
# mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
```

4. Initialize the host UIO driver:

```
# modprobe uio
# insmod $RTE_SDK/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko
```

5. Locate the target NICs to be used for the test, attach to the UIO driver, and check:

```
# lspci -nn |grep Ethernet
03:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-
Gigabit X540-AT2 [8086:1528] (rev 01)
03:00.1 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-
Gigabit X540-AT2 [8086:1528] (rev 01)
06:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)
06:00.1 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)
06:00.2 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)
06:00.3 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)

# $RTE_SDK/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.0
# $RTE_SDK/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.1
# $RTE_SDK/tools/dpdk_nic_bind.py --status
```

6. Use the following command line, to run the DPDK L3 forwarding application on the compute node host:

```
# cd $RTE_SDK/examples/l3fwd
# ./build/l3fwd -c 0x6 -n 4 --socket-mem 1024,0 -- -p 0x3 -
config="(0,0,1),(1,0,2)" &
```

Note: The `-c` option enables cores (1,2) to run the application. The `--socket-mem` option just allocates 1 GB of the hugepage memory from target NUMA node 0 and zero memory from NUMA node 1. The `-p` option is the hexadecimal bit mask of the ports to configure. The `--config (port, queue, lcore)` option determines which queues from which ports are mapped to which cores.

7. When l3fwd is executed, some information about the forwarding operation is displayed during startup:

```
...
Initializing port 0 ... Creating queues: nb_rxq=1 nb_txq=2...
Address:68:05:CA:29:A4:C8, Allocated mbuf pool on socket 0
LPM: Adding route 0x01010100 / 24 (0)
LPM: Adding route 0x02010100 / 24 (1)
LPM: Adding route IPV6 / 48 (0)
LPM: Adding route IPV6 / 48 (1)
...
Initializing port 1 ... Creating queues: nb_rxq=1 nb_txq=2...
Address:68:05:CA:29:A4:C9, txq=1,0,0
```

The Ethernet addresses for port 0 and port 1 are displayed, which can be useful for setting

up the test equipment. For example, these values are used to set the IPv4 Gateway MAC addresses for a Spirent test equipment for the ports under test. The forwarding route 0x01010100 / 24 gives an IPv4 address range for the port 0 test interface. Setting the test equipment IPv4 Address to 1.1.1.5 has worked for port 0 and 2.1.1.5 port 1.

8. Use the following command line to run the DPDK L3 forwarding application for 4 ports on the compute node host by adding ports:

```
# $RTE_SDK/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.2
# $RTE_SDK/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.3

# cd /usr/src/dpdk/examples/l3fwd
# ./build/l3fwd -c 1e -n 4 --socket-mem 1024,0 -- -p 0xf --
config="(0,0,1),(1,0,2),(2,0,3),(3,0,4)" &
```

E.3 Running the DPDK L2 Forwarding Application

The L2 forward application modifies the packet data by changing the Ethernet address in the outgoing buffer which might impact performance. L3fwd program forwards the packet without modifying it.

Build DPDK and l2fwd as described in Appendix E.1. Building DPDK and L3/L2 Forwarding Applications. Set up the host boot line for 1 GB and 2 MB hugepage memory and isolcpus as follows:

1. Initialize host hugepage memory after reboot:

```
# mount -t hugetlbfs nodev /dev/hugepages
# mkdir /dev/hugepages_2mb
# mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
```

2. Initialize the host userspace I/O (UIO) driver:

```
# modprobe uio
# insmod $RTE_SDK/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko
```

3. Locate target NICs to be used for test (assuming `06:00.0` and `06:00.01` are target), attach to UIO driver, and check:

```
# lspci -nn |grep Ethernet
03:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-
Gigabit X540-AT2 [8086:1528] (rev 01)
03:00.1 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-
Gigabit X540-AT2 [8086:1528] (rev 01)
06:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)
06:00.1 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)
06:00.2 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)
06:00.3 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)

# $RTE_SDK/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.0
# $RTE_SDK/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.1
# $RTE_SDK/tools/dpdk_nic_bind.py -status
```

4. To run the DPDK l2 forwarding application on the compute node host, use the following command line:

```
# cd $RTE_SDK/examples/l2fwd
# ./build/l2fwd -c 6 -n 4 --socket-mem 1024,0 -- -p 3 &
```

Note: The `-c` option enables cores (1, 2) to run the application. The `--socket-mem` option allocates 1 GB of the hugepage memory from target NUMA node 0 and zero memory from NUMA node 1. The `-p` option is the hexadecimal bit mask of the ports to configure.

5. Use the following command to run the DPDK l2 forwarding application for 4 ports on the compute node host by adding ports:

```
# $RTE_SDK/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.2
# $RTE_SDK/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.3

# cd /usr/src/dpdk/examples/l2fwd
# ./build/l2fwd -c 1e -n 4 --socket-mem 1024,0 -- -p 0f &
```

Appendix F Open vSwitch

F.1 Building DPDK for OVS

Build DPDK for Open vSwitch (OVS) as explained in this section.

F.1.1 Getting OVS Git Source Repository

1. Make sure the DPDK is currently built for the OVS build, if not, change as documented and rebuild DPDK.
2. Get the OVS git repository:

   ```
   # cd /usr/src
   # git clone https://github.com/openvswitch/ovs.git
   ```

3. Move into the new OVS git repository:

   ```
   # cd /usr/src/ovs
   ```

4. Assume that the latest commit is the target commit. Do one of the following:
   a. Create a new work branch (`test_branch`) off the current master branch to do testing and for applying the user vhost patch:

      ```
      # git checkout –b test_branch
      ```

   b. Check out the specific commit used in OVS testing of this document:

      ```
      # git checkout –b test_branch 7cc398cb8561a16ae3be5ffc687be5620981d619
      ```

The user vhost interface patch needs to be applied for OVS user vhost tests per the following section.

F.1.2 Getting OVS DPDK-netdev User Space vHost Patch

Not all the user vhost DPDK patches may have been upstreamed into the OVS source code. The patches used can be obtained from OVS project (March 5, 2015):

http://openvswitch.org/pipermail/dev/2015-March/052061.html

`[ovs-dev] [PATCH v8] netdev-dpdk: add dpdk vhost-cuse ports`

The update code can be cut from the message and saved into a text file to create the patch. The patch can be named whatever is convenient and placed in the directory just above the local OVS git repository directory *(/usr/src)*. Our example, patch name is `Patch_netdev_add_dpdk_vhost-cuse_ports`.

If copied using a Windows machine, the text file needs to be converted into a Linux text-type file from the DOS text-file format.

F.1.3 Applying the OVS DPDK-netdev User Space vHost Patch

Before applying the patch, it is preferable to use your own git branch to prevent source tree switching issues. A new branch creation is shown in the checkout section for getting the OVS repository. With the patch placed in the */usr/src* directory, the following code applies the patch named in the previous section:

```
cd /usr/src/ovs
patch -p1 < ../ Patch_netdev_add_dpdk_vhost-cuse_ports
patching file INSTALL.DPDK.md
patching file Makefile.am
patching file lib/automake.mk
patching file lib/netdev-dpdk.c
patching file lib/netdev.c
patching file utilities/automake.mk
patching file utilities/qemu-wrap.py
patching file vswitchd/ovs-vswitchd.c
```

Assuming that the patch applies correctly, changes can be checked into the git repository for the test branch. This enables easy switching between different OVS commit versions in different git branches. The patch can be applied subsequently to a later point in the master code repository branch on a new branch.

F.1.4 Building OVS with DPDK-netdev

The DPDK needs to be built for OVS as described in Appendix F.1, Building DPDK for OVS. The following assumes DPDK is from the git repository in */usr/src/dpdk.* Adjust the DPDK path, if from another DPDK build directory.

```
# cd /usr/src/ovs
# make clean
# export DPDK_BUILD=/usr/src/dpdk/x86_64-ivshmem-linuxapp-gcc
# ./boot.sh
# ./configure --with-dpdk=$DPDK_BUILD
# make
```

F.2 OVS Throughput Tests

F.2.1 OVS Base Setup for Tests

Build OVS as described in section F.1.4, Building OVS with DPDK-netdev, reboot the system, and check the kernel boot line for 1 GB hugepage, iosolcpu setting for target cores (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39 for 14 core CPU with hyperthreading enabled), and the IOMMU disabled:

```
# dmesg |grep command
[    0.000000] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-3.18.5-201.fc21.x86_64
root=UUID=cc17be46-707c-414a-8c16-11ecda5905be ro default_hugepagesz=1GB
hugepagesz=1GB hugepages=16 hugepagesz=2M hugepages=2048
isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,28,29,30,31,32,33,34,35,36,37,38,39 rhgb quiet
```

1. Load the OVS kernel module and check for it being present:

```
# modprobe openvswitch
# lsmod |grep open
openvswitch            75016  0
geneve                 13166  1 openvswitch
gre                    13535  1 openvswitch
libcrc32c              12603  1 openvswitch
vxlan                  36832  2 i40e,openvswitch
```

2. Load the kernel UIO driver and DPDK UIO driver:

```
# modprobe uio
# insmod /usr/src/dpdk/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko
```

3. Load the cuse and fuse kernel modules:

```
# modprobe cuse
# modprobe fuse
```

4. Remove the vhost-net kernel module and delete any associated data:

```
# rmmod vhost-net
# rm -f /dev/vhost-net
```

5. Load the event link kernel module for the user side vhost:

```
# insmod /usr/src/dpdk/lib/librte_vhost/eventfd_link/eventfd_link.ko
```

6. Find the target Ethernet interfaces:

```
# lspci -nn |grep Ethernet
03:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-
Gigabit X540-AT2 [8086:1528] (rev 01)
03:00.1 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-
Gigabit X540-AT2 [8086:1528] (rev 01)
06:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)
06:00.1 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)
06:00.2 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
for 10GbE SFP+ [8086:1572] (rev 01)
06:00.3 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710
```

```
for 10GbE SFP+ [8086:1572] (rev 01)
```

7. Bind Ethernet interfaces to the UIO interface (# config xxxx down):

```
# /usr/src/dpdk/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.0
# /usr/src/dpdk/tools/dpdk_nic_bind.py --bind=igb_uio 06:00.1
# /usr/src/dpdk/tools/dpdk_nic_bind.py --status #
```

8. Mount the hugepage file systems for both 1 GB and 2 MB hugepages:

```
# mount -t hugetlbfs nodev /dev/hugepages
# mkdir /dev/hugepages_2mb
# mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
# cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
# echo 2048 > /sys/devices/system/node/node0/hugepages/hugepages-
2048kB/nr_hugepages
```

9. Terminate OVS and delete the previous OVS database (does not need to be done, if valid DB without a new OVS build):

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

10. Set up directories for the new OVS database (does not need to be done, if valid DB without a new OVS build):

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

11. Initialize the new OVS database (does not need to be done, if valid DB without a new OVS build):

```
# cd /usr/src/ovs
# ./ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db \
./vswitchd/vswitch.ovsschema
```

12. Start the OVS database server:

```
# cd /usr/src/ovs
# ./ovsdb/ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
--remote=db:Open_vSwitch,Open_vSwitch,manager_options \
--private-key=db:Open_vSwitch,SSL,private_\
key --certificate=db:Open_vSwitch,SSL,certificate \
--bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert --pidfile --detach
```

13. Initialize the OVS database (only after creating a new DB):

```
# cd /usr/src/ovs
# ./utilities/ovs-vsctl --no-wait init
```

14. Start OVS with DPDK portion using 2 GB of node 0 memory:

```
# cd /usr/src/ovs
# /vswitchd/ovs-vswitchd --dpdk -c 0x4 -n 4 --socket-mem 2048,0 --
unix:/usr/local/var/run/openvswitch/db.sock --pidfile --detach
```

15. Set the default flow timeout to 30 seconds:

```
# cd /usr/src/ovs/utilities
# ./ ovs-vsctl set o . other_config:max-idle=30000
```

16. Set the default OVS PMD thread usage:

```
# cd /usr/src/ovs/utilities
# ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=2
```

17. Create the OVS DPDK bridge and add the two physical NICs. This needs to be done sequentially one line at a time (or have a delay between each in a script) so that the port order is assigned sequentially. If previous ports were created in a different order or other ports were created, the port IDs of the ports might not be correct for the test. Either the flow set port IDs need to be changed in the OVS flow commands or the OVS tasks need to be stopped and the previous OVS database deleted, OVS tasks restarted, and new database created.

```
# cd /usr/src/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
```

18. Check OVS task affinitization (use your system PIDs):

```
# ps -e |grep ovs
 15544 ?        00:00:00 ovsdb-server
 15901 ?        01:38:16 ovs-vswitchd

# top -p 15901 -H -d1

Threads:  33 total,   1 running,  32 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3.6 us,  0.0 sy,  0.0 ni, 96.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 65841244 total, 47628324 free, 17427804 used,   785116 buff/cache
KiB Swap:  8388604 total,  8388604 free,        0 used. 48081896 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 15966 root      20   0 12.818g   7232   4644 R 99.9  0.0  21:06.58 pmd59
 15901 root      20   0 12.818g   7232   4644 S  3.0  0.0   0:08.85 ovs-
vswitchd
 15902 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:12.28
dpdk_watchdog3
 15903 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.02
cuse_thread2
 15904 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 urcu1
 15938 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler61
 15939 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler60
 15940 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler58
 15941 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler57
 15942 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler56
 15943 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler55
 15944 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler33
 15945 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler34
 15946 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler35
 15947 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler36
 15948 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler37
 15949 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler38
 15950 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler39
 15951 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler40
 15952 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler41
```

```
 15953 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler42
 15954 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler43
 15955 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler44
 15956 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler45
 15957 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.00 handler46
 15958 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.24
revalidator47
 15959 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.09
revalidator48
 15960 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.08
revalidator49
 15961 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.07
revalidator50
 15962 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.09
revalidator51
 15963 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.09
revalidator52
 15964 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.08
revalidator53
 15965 root      20   0 12.818g   7232   4644 S  0.0  0.0   0:00.08
revalidator54

# taskset -p 15901
pid 15901's current affinity mask: 4
# taskset -p 15902
pid 15902's current affinity mask: 4
# taskset -p 15903
pid 15903's current affinity mask: 4
# taskset -p 15904
pid 15904's current affinity mask: 4
# taskset -p 15938
pid 15938's current affinity mask: 4
# taskset -p 15939
pid 15939's current affinity mask: 4
# taskset -p 15940
pid 15940's current affinity mask: 4
# taskset -p 15941
pid 15941's current affinity mask: 4
# taskset -p 15942
pid 15942's current affinity mask: 4
# taskset -p 15943
pid 15943's current affinity mask: 4
# taskset -p 15944
pid 15944's current affinity mask: 4
# taskset -p 15945
pid 15945's current affinity mask: 4
# taskset -p 15946
pid 15946's current affinity mask: 4
# taskset -p 15947
pid 15947's current affinity mask: 4
# taskset -p 15948
pid 15948's current affinity mask: 4
# taskset -p 15949
```

```
pid 15949's current affinity mask: 4
# taskset -p 15950
pid 15950's current affinity mask: 4
# taskset -p 15951
pid 15951's current affinity mask: 4
# taskset -p 15952
pid 15952's current affinity mask: 4
# taskset -p 15953
pid 15953's current affinity mask: 4
# taskset -p 15954
pid 15954's current affinity mask: 4
# taskset -p 15955
pid 15955's current affinity mask: 4
# taskset -p 15956
pid 15956's current affinity mask: 4
# taskset -p 15957
pid 15957's current affinity mask: 4
# taskset -p 15958
pid 15958's current affinity mask: 4
# taskset -p 15959
pid 15959's current affinity mask: 4
# taskset -p 15960
pid 15960's current affinity mask: 4
# taskset -p 15961
pid 15961's current affinity mask: 4
# taskset -p 15962
pid 15962's current affinity mask: 4
# taskset -p 15963
pid 15963's current affinity mask: 4
# taskset -p 15964
pid 15964's current affinity mask: 4
# taskset -p 15965
pid 15965's current affinity mask: 4
# taskset -p 15966
pid 15966's current affinity mask: 2
```

Note: OVS task affiliation will change in the future.

F.2.2 OVS PHY-PHY Throughput Tests

After the OVS is started, running the port-to-port test first requires setting flows and doing a few checks:

1. Set up OVS for the test as described in Appendix F.2.1, OVS Base Setup for Tests.
2. Make sure the OVS DPDK bridge contains the two physical NICs:

```
# cd /usr/src/ovs
# ./utilities/ovs-vsctl show
157fa72b-57a3-4a88-86aa-e6d6a82458a1
    Bridge "br0"
        Port "dpdk1"
```

```
            Interface "dpdk1"
                type: dpdk
        Port "dpdk0"
            Interface "dpdk0"
                type: dpdk
        Port "br0"
            Interface "br0"
                type: internal
```

Note: UUID of 157fa72b-57a3-4a88-86aa-e6d6a82458a1 for this OVS instance, and is different for each instance.

3. Assuming the test equipment IPv4 endpoint of `1.1.1.1` for port 1 and `5.1.1.1` for port 2 (port IDs are not displayed in the `show` command above), the flow rules need to be adjusted for other endpoint IP addresses. Set the OVS port-to-port flow rules for the NIC 0 (port 1) endpoint IP address to `1.1.1.1` and NIC 1 (port 2) endpoint IP address to `5.1.1.1`, using the following script:

```
# cat ovs_flow_ports.sh
#! /bin/sh
# Move to command directory
cd /usr/src/ovs/utilities/
# Clear current flows
./ovs-ofctl del-flows br0
# Add Flow for port 0 to port 1 and port 1 to port 0
./ovs-ofctl add-flow br0
in_port=1,dl_type=0x800,nw_src=1.1.1.1,nw_dst=5.1.1.1,idle_timeout=0,action=out
put:2
./ovs-ofctl add-flow br0
in_port=2,dl_type=0x800,nw_src=5.1.1.1,nw_dst=1.1.1.1,idle_timeout=0,action=out
put:1
```

   Apply the script:

```
# ./ovs_flow_ports.sh
```

4. Check the host CPU load to verify the OVS PMD is operating (this is for the 1 OVS PMD thread case):

```
# top (1)
top - 18:29:24 up 37 min, 4 users, load average: 0.86, 0.72, 0.63
Tasks: 275 total, 1 running, 274 sleeping, 0 stopped, 0 zombie
%Cpu0 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu1 :100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
...
```

   This shows the OVS 1 PMD thread case where CPU core 1 is being used.

5. Affinitize DPDK PDM task to CPU core 2 (note that core 4 is used by non-PMD OVS threads at this time):

```
# cd /usr/src/ovs/utilities
```

   For 1 PMD core test (cores 1) and must be compatible to other cores usage:

```
# ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=2
```

   For 2 PMD core test (cores 1 and 3) and must be compatible to other cores usage:

```
# ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0a
```

For 4 PMD core test (cores 1, 3, 4, and 5):

```
# ./ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=3a
```

F.5.3 Starting up the VM for (OVS User Space vHost) Throughput Test

Verify host kernel settings and setup and run OVS in the same way as with OVS PHY-PHY above. The current User Space vHost under test will only work correctly with one OVS PMD thread. The current user space vhost under test will only work correctly with one OVS PMD thread. Running the user space vhost interface with more than one PMD thread causes poor throughput, which can only be recovered by terminating the VM and restarting `ovs_vswitched`. This should be corrected in upcoming user space vhost improvements.

1. Set up OVS for the test as described in Appendix F.2.1, OVS Base Setup for Tests, and Appendix F.2.2, OVS PHY-PHY Throughput Tests excluding the load of the OVS flow rules.
2. Verify QEMU version for Fedora 21:

   ```
   # qemu-system-x86_64 -version
   QEMU emulator version 2.1.3 (qemu-2.1.3-2.fc21), Copyright (c) 2003-2008
   Fabrice Bellard
   ```

   Note: The test can probably work on multiple versions of QEMU, but this has not been verified.
3. Add the user-side vhost ports to OVS DPDK bridge br0, they need to be present for a VM to start:

   ```
   # cd /usr/src/ovs
   # ./utilities/ovs-vsctl add-port br0 dpdkvhost0 -- set Interface dpdkvhost0
   type=dpdkvhost
   # ./utilities/ovs-vsctl add-port br0 dpdkvhost1 -- set Interface dpdkvhost1
   type=dpdkvhost
   ```

   Note: The ovs-vswitchd needs to be running and each is added in the order after the database has been cleared to establish dpdk0 as port 1, dpdk1 as port 2, dpdkvhost0 as port 3, and dpdkvhost1 as port 4 for the example flow rules to be correct. If the database was not cleared, previous port ID assignments remain and the port IDs are not always in the expected order. In that case, the following needs to be done in this order: the flow commands adjusted or the OVS terminated, OVS database deleted, OVS database created, OVS restarted, and the bridge created.
4. Verify the vhost ports **dpdkvhost0** and **dpdkvhost0** (each bridge has a UUID):

   ```
   # ./utilities/ovs-vsctl show
   157fa72b-57a3-4a88-86aa-e6d6a82458a1
       Bridge "br0"
           Port "dpdk1"
               Interface "dpdk1"
                   type: dpdk
           Port "dpdkvhost1"
               Interface "dpdkvhost1"
                   type: dpdkvhost
           Port "dpdk0"
               Interface "dpdk0"
   ```

```
                type: dpdk
        Port "br0"
            Interface "br0"
                type: internal
        Port "dpdkvhost0"
            Interface "dpdkvhost0"
                type: dpdkvhost
```

Note: The UUID for this OVS database instance is 157fa72b-57a3-4a88-86aa-e6d6a82458a1.

5. Check the system for sufficient 1 GB hugepage memory to run the target VMs:

```
# cat /proc/meminfo
...
HugePages_Total:      16
HugePages_Free:        14
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:    1048576 kB
...
```

For the 2-socket system, half of the 1 GB hugepages are from NUMA node 0 and half from NUMA node 1 (or with 16 total); eight hugepages are from each NUMA node. In this case, two hugepages from NUMA node 0 are being used by ovs-vswitchd, leaving six 1 GB NUMA 0 pages for the VMs. The minimum number of 1 GB hugepages that can be used that will allow one 1 GB page to be available in the VM requires starting the VM with a minimum of three 1 GB hugepages. For the available memory, two VMs (3 GB memory each) of this type can be executed using NUMA node 0 memory.

The following is an example of a VM startup script (located in */vm/vm_ctl*, but it can be in another directory):

```
# cd /vm/vm_ctl
# cat Fed21_VM_vhost.sh

#!/bin/sh

vm=/vm/Fed21-mp.qcow2
vm_name=Fed21_VM_PCI
vnc=14
n1=tap55
bra=br-mgt
dn_scrp_a=/vm/vm_ctl/br-mgt-ifdown
mac1=00:00:14:42:04:29

if [ ! -f $vm ];
then
    echo "VM $vm not found!"
else
    echo "VM $vm started! VNC: $vnc, net0: $n1, net1: $n2"
    tunctl -t $n1
    brctl addif $bra $n1
    ifconfig $n1 0.0.0.0 up
    taskset 38 qemu-system-x86_64 \
-m 3072 -smp 3 -cpu host -hda $vm -boot c -enable-kvm \
-pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait \
```

```
-object memory-backend-file,id=mem,size=3072M,mem-path=/dev/hugepages,share=on
\
-numa node,memdev=mem -mem-prealloc \
-net nic,model=virtio,netdev=eth0,macaddr=$mac1 \
-netdev tap,ifname=$n1,id=eth0,script=no,downscript=$dn_scrp_a \
-netdev type=tap,id=net1,script=no,downscript=no,ifname=dpdkvhost0,vhost=on \
-device virtio-net-pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,\
guest_tso4=off,guest_tso6=off,guest_ecn=off \
-netdev type=tap,id=net2,script=no,downscript=no,ifname=dpdkvhost1,vhost=on \
-device virtio-net-pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,\
guest_tso4=off,guest_tso6=off,guest_ecn=off \
-vnc :$vnc -name $vm_name &
Fi

# cat br-mgt-ifdown
#!/bin/sh

bridge='br-mgt'
/sbin/ifconfig $1 0.0.0.0 down
brctl delif  ${bridge} $1
```

Notes:

1. QEMU is started on cores 3, 4,5 (0x038) to prevent interfering with other cores and force all cores to run on CPU socket 0. Therefore, all allocations will be on NUMA node 0 to prevent a NUMA node 1 memory allocation.
2. The section "-`object memory-backend-file,id=mem,size=3072M,mem-path=/dev/ hugepages,share=on -numa node,memdev=mem -mem-prealloc`" allocates the hugepage memory and must be sized the same as the "`-m`" argument. There must be a minimum size of 3 GB to use the 1 GB hugepage in the VM.
3. The NICs in the VIM were ordered the same as listed in the QEMU command line. "`- net nic,model=virtio,netdev=eth0,macaddr=$mac1 \-netdev tap, ifname=$n1,id=eth0,script=no,downscript=$dn_scrp_a`" is the first NIC and uses the standard virtio protocol stack for general control of the VM using an ssh terminal. Note the vhost was not used for this test case because it interfered with the way the user side vhost was set up.
4. The second NIC argument "`-netdev type=tap,id=net1,script=no, downscript=no,ifname=dpdkvhost0,vhost=on -device virtio-net-pci, netdev= net1, mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off, guest_tso6=off,guest_ecn=off`" assigns the user side vhost port "`dpdkvhost0`", by name to NIC 2.
5. The third NIC argument assigns the user side vhost port "`dpdkvhost1`" by name to QEMU NIC device 3.
6. Three virtual processors (`-smp 3`) were used because the same VM was used for other tests. The user space vhost test uses only one real-time process, so it can be started with two virtual processor cores. A minimum of two vCPUs were used because at least 1 vCPU must be available that is not running the real-time process to run Linux OS.
7. The VM uses DPDK, which requires the use of a recent CPU core, such as the host has. Use host CPU core (-`cpu host`).

6. Start the Fedora 21 VM.

```
# ./Fed21_VM_vhost.sh
```

7. Apply the vhost to VM vhost flow operations for a single VM:

```
# ./ovs_flow_vm_vhost.sh
# cat ovs_flow_vm_vhost.sh
#! /bin/sh
# Move to command directory cd /usr/src/ovs/utilities/
# Clear current flows
./ovs-ofctl del-flows br0
# Add Flow for port 0 (16) to port 1 (17)
./ovs-ofctl add-flow br0
in_port=2,dl_type=0x800,nw_dst=5.1.1.1,idle_timeout=0,action=output:4
./ovs-ofctl add-flow br0
in_port=1,dl_type=0x800,nw_src=5.1.1.1,idle_timeout=0,action=output:3
./ovs-ofctl add-flow br0
in_port=4,dl_type=0x800,nw_src=5.1.1.1,idle_timeout=0,action=output:2
./ovs-ofctl add-flow br0
in_port=3,dl_type=0x800,nw_dst=5.1.1.1,idle_timeout=0,action=output:1
```

   In this case, the endpoint port 2 has fixed IP endpoint `5.1.1.1`, while the port 1 endpoint has multiple endpoints IPs starting at IP `1.1.1.1` and incrementing up to half the bidirectional flows of the test. This allows a large number of flows to be set using a small number of inexact flow commands. The `nw_src` and `nd_dst` can also be deleted and would result in the same flows, but without any IP matches in the inexact flow rules.

F.5.3.1 Operating the VM for the (OVS User Space vHost) Throughput Test

The following operations are done in the VM:

1. The VM kernel boot line needs to have 1 GB of hugepage memory, a moderate number of 2 MB pages configured, and at least one isolated vCPU. This VM kernel was configured to support all the tests in this document. The 2 MB hugepages are used in this test. Check for the correct bootline parameters, change if needed, and then reboot.

```
# dmesg |grep command
[    0.000000] Kernel command line: BOOT_IMAGE=/vmlinuz-3.18.7-200.fc21.x86_64
root=/dev/mapper/fedora--server-root ro default_hugepagesz=1GB hugepagesz=1GB
hugepages=1 hugepagesz=2M hugepages=256 isolcpus=1,2 rd.lvm.lv=fedora-
server/root rd.lvm.lv=fedora-server/swap rhgb quiet
```

2. Get the DPDK package (same as host):

```
# cd /usr/src
# git clone git://dpdk.org/dpdk
# cd /usr/src/dpdk
```

3. Check out the target DPDK 1.8.0 version:

```
# git checkout -b test_v1.8.0 v1.8.0
```

4. The application used for the test is a DPDK application; therefore, so DPDK install must be for a DPDK application:

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

5. The test-pmd application's input and output queue sizes are increased to 2K for better throughput performance. Use th editor or create and apply a patch from the output text below:

```
# cd /usr/src/dpdk/app/test-pmd
# vim testpmd.c
# git diff -M -C
diff --git a/app/test-pmd/testpmd.c b/app/test-pmd/testpmd.c
index 8c69756..c625f2e 100644
--- a/app/test-pmd/testpmd.c
+++ b/app/test-pmd/testpmd.c
@@ -194,8 +194,8 @@ queueid_t nb_txq = 1; /**< Number of TX queues per port. */
 /*
  * Configurable number of RX/TX ring descriptors.
  */
-#define RTE_TEST_RX_DESC_DEFAULT 128
-#define RTE_TEST_TX_DESC_DEFAULT 512
+#define RTE_TEST_RX_DESC_DEFAULT 2048
+#define RTE_TEST_TX_DESC_DEFAULT 2048
 uint16_t nb_rxd = RTE_TEST_RX_DESC_DEFAULT; /**< Number of RX descriptors. */
 uint16_t nb_txd = RTE_TEST_TX_DESC_DEFAULT; /**< Number of TX descriptors. */
Build the test-pmd application.
# export RTE_SDK=/usr/src/dpdk
# export RTE_TARGET=x86_64-ivshmem-linuxapp-gcc
# make
  CC testpmd.o
  CC parameters.o
  CC cmdline.o
  CC config.o
  CC iofwd.o
  CC macfwd.o
  CC macfwd-retry.o
  CC macswap.o
  CC flowgen.o
  CC rxonly.o
  CC txonly.o
  CC csumonly.o
  CC icmpecho.o
  CC mempool_anon.o
  LD testpmd
  INSTALL-APP testpmd
  INSTALL-MAP testpmd.map
```

6. If the testpmd was already built, such as a VM restart, start here. Set up a hugepage filesystem:

```
# mkdir -p /mnt/hugepages
# mount -t hugetlbfs hugetlbfs /mnt/hugepages
# mkdir /dev/hugepages_2mb
# mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
```

7. Load the `UIO` kernel modules:

```
# modprobe uio
# insmod /usr/src/dpdk/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko
```

8. Find the passed user-side vhost network devices. The device order is in the order listed in the QEMU startup command line. The first network device is the management network, the second and third network device are the user side vhost network devices:

```
# lspci -nn
00:00.0 Host bridge [0600]: Intel Corporation 440FX - 82441FX PMC [Natoma]
[8086:1237] (rev 02)
00:01.0 ISA bridge [0601]: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton
II] [8086:7000]
00:01.1 IDE interface [0101]: Intel Corporation 82371SB PIIX3 IDE
[Natoma/Triton II] [8086:7010]
00:01.3 Bridge [0680]: Intel Corporation 82371AB/EB/MB PIIX4 ACPI [8086:7113]
(rev 03)
00:02.0 VGA compatible controller [0300]: Cirrus Logic GD 5446 [1013:00b8]
00:03.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
00:04.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
00:05.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
```

9. Bind the user side vhost network devices to the igb_uio device driver:

```
# /usr/src/dpdk/tools/dpdk_nic_bind.py -b igb_uio 0000:00:04.0
# /usr/src/dpdk/tools/dpdk_nic_bind.py -b igb_uio 0000:00:05.0
```

10. Check the bind status to make sure PCI 0000:00:04.0 and 0000:00:05.0 are bound to the igb_uio device driver:

```
# /usr/src/dpdk/tools/dpdk_nic_bind.py --status

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

Network devices using kernel driver
===================================
0000:00:03.0 'Virtio network device' if= drv=virtio-pci
unused=virtio_pci,igb_uio

Other network devices
=====================
<none>
```

11. Run the test-pmd application (running in screen allows terminal termination and reconnection later while the application is running; otherwise, if the terminal is shut down, testpmd terminates):

```
# screen -S test-pmd
# cd /usr/src/dpdk/app/test-pmd/
# ./testpmd -c 0x3 -n 4 --socket-mem 128 -- --burst=64 -i --txqflags=0xf00
EAL: Detected lcore 0 as core 0 on socket 0
EAL: Detected lcore 1 as core 0 on socket 0
```

```
EAL: Detected lcore 2 as core 0 on socket 0
EAL: Support maximum 128 logical core(s) by configuration.
EAL: Detected 3 lcore(s)
EAL:   unsupported IOMMU type!
EAL: VFIO support could not be initialized
EAL: Searching for IVSHMEM devices...
EAL: No IVSHMEM configuration found!
EAL: Setting up memory...
EAL: Ask a virtual area of 0x40000000 bytes
EAL: Virtual area found at 0x7f25c0000000 (size = 0x40000000)
EAL: Ask a virtual area of 0x200000 bytes
EAL: Virtual area found at 0x7f2684800000 (size = 0x200000)
EAL: Ask a virtual area of 0x1200000 bytes
EAL: Virtual area found at 0x7f2683400000 (size = 0x1200000)
EAL: Ask a virtual area of 0x200000 bytes
EAL: Virtual area found at 0x7f2684e00000 (size = 0x200000)
EAL: Ask a virtual area of 0xe800000 bytes
EAL: Virtual area found at 0x7f2674a00000 (size = 0xe800000)
EAL: Ask a virtual area of 0x200000 bytes
EAL: Virtual area found at 0x7f2674600000 (size = 0x200000)
EAL: Ask a virtual area of 0xfc00000 bytes
EAL: Virtual area found at 0x7f2664800000 (size = 0xfc00000)
EAL: Ask a virtual area of 0x200000 bytes
EAL: Virtual area found at 0x7f26a4800000 (size = 0x200000)
EAL: Ask a virtual area of 0x200000 bytes
EAL: Virtual area found at 0x7f26a4400000 (size = 0x200000)
EAL: Requesting 64 pages of size 2MB from socket 0
EAL: TSC frequency is ~2593993 KHz
EAL: WARNING: cpu flags constant_tsc=yes nonstop_tsc=no -> using unreliable
clock cycles !
EAL: Master core 0 is ready (tid=a590f840)
PMD: ENICPMD trace: rte_enic_pmd_init
EAL: Core 1 is ready (tid=a4212700)
EAL: PCI device 0000:00:03.0 on NUMA socket -1
EAL:   probe driver: 1af4:1000 rte_virtio_pmd
EAL:   0000:00:03.0 not managed by UIO driver, skipping
EAL: PCI device 0000:00:04.0 on NUMA socket -1
EAL:   probe driver: 1af4:1000 rte_virtio_pmd
EAL:   PCI memory mapped at 0x7f2685000000
EAL: PCI device 0000:00:05.0 on NUMA socket -1
EAL:   probe driver: 1af4:1000 rte_virtio_pmd
EAL:   PCI memory mapped at 0x7f2685001000
Interactive-mode selected
Configuring Port 0 (socket 0)
Port 0: 00:00:00:00:00:01
Configuring Port 1 (socket 0)
Port 1: 00:00:00:00:00:02
Checking link statuses...
Port 0 Link Up - speed 10000 Mbps - full-duplex
Port 1 Link Up - speed 10000 Mbps - full-duplex
Done
testpmd>
```

12. In the application, enter `fwd` and `mac_retry` commands to set up the operation:

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

13. Start the PMD forwarding operation:

```
testpmd> start
  mac_retry packet forwarding - CRC stripping disabled - packets/burst=64
  nb forwarding cores=1 - nb forwarding ports=2
  RX queues=1 - RX desc=2048 - RX free threshold=32
  RX threshold registers: pthresh=8 hthresh=8 wthresh=0
  TX queues=1 - TX desc=2048 - TX free threshold=0
  TX threshold registers: pthresh=32 hthresh=0 wthresh=0
  TX RS bit threshold=0 - TXQ flags=0xf00
testpmd>
```

14. Back on host, affinitize the vCPU 1 to an available target real-time CPU core. Review the QEMU tasks:

```
# ps -eLF|grep qemu
root      16980      1  16980  0    6 1236566 40484 3 10:47 pts/0    00:00:01
qemu-system-x86_64 -m 4096 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
enable-kvm -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
object memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -
numa node,memdev=mem -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -netdev
type=tap,id=net1,script=no,downscript=no,ifname=dpdkvhost0,vhost=on -device
virtio-net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -netdev
type=tap,id=net2,script=no,downscript=no,ifname=dpdkvhost1,vhost=on -device
virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -vnc :14 -name Fed21_VM_PCI
root      16980      1  16986  4    6 1236566 40484 3 10:47 pts/0    00:00:13
qemu-system-x86_64 -m 4096 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
enable-kvm -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
object memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -
numa node,memdev=mem -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -netdev
type=tap,id=net1,script=no,downscript=no,ifname=dpdkvhost0,vhost=on -device
virtio-net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -netdev
type=tap,id=net2,script=no,downscript=no,ifname=dpdkvhost1,vhost=on -device
virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -vnc :14 -name Fed21_VM_PCI
root      16980      1  16988 19    6 1236566 40484 3 10:47 pts/0    00:00:55
qemu-system-x86_64 -m 4096 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
enable-kvm -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
object memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -
numa node,memdev=mem -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -netdev
type=tap,id=net1,script=no,downscript=no,ifname=dpdkvhost0,vhost=on -device
virtio-net-
```

```
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -netdev
type=tap,id=net2,script=no,downscript=no,ifname=dpdkvhost1,vhost=on -device
virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -vnc :14 -name Fed21_VM_PCI
root      16980      1  16989  0    6 1236566 40484 3 10:47 pts/0    00:00:01
qemu-system-x86_64 -m 4096 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
enable-kvm -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
object memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -
numa node,memdev=mem -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -netdev
type=tap,id=net1,script=no,downscript=no,ifname=dpdkvhost0,vhost=on -device
virtio-net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -netdev
type=tap,id=net2,script=no,downscript=no,ifname=dpdkvhost1,vhost=on -device
virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -vnc :14 -name Fed21_VM_PCI
root      16980      1  16991  0    6 1236566 40484 3 10:47 pts/0    00:00:00
qemu-system-x86_64 -m 4096 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
enable-kvm -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
object memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -
numa node,memdev=mem -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -netdev
type=tap,id=net1,script=no,downscript=no,ifname=dpdkvhost0,vhost=on -device
virtio-net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -netdev
type=tap,id=net2,script=no,downscript=no,ifname=dpdkvhost1,vhost=on -device
virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -vnc :14 -name Fed21_VM_PCI
root      16980      1  16998  0    6 1236566 40484 3 10:51 pts/0    00:00:00
qemu-system-x86_64 -m 4096 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
enable-kvm -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
object memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on -
numa node,memdev=mem -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -netdev
type=tap,id=net1,script=no,downscript=no,ifname=dpdkvhost0,vhost=on -device
virtio-net-
pci,netdev=net1,mac=00:00:00:00:00:01,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -netdev
type=tap,id=net2,script=no,downscript=no,ifname=dpdkvhost1,vhost=on -device
virtio-net-
pci,netdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,guest_tso
6=off,guest_ecn=off -vnc :14 -name Fed21_VM_PCI
```

15. In this case, we see that QEMU thread 16988 is accumulating CPU runtime, which is vCPU 1. We change the affinitization of vCPU 1 and vCPU 2 to the target assigned cores reserved for the real-time operation; vCPU 0 is left on the previous affinitized startup core set.

```
# # taskset -p 40 16988
pid 16988's current affinity mask: 38
pid 16988's new affinity mask: 40
[root@SRT1-3 vm_ctl]# taskset -p 80 16989
```

```
pid 16989's current affinity mask: 38
pid 16989's new affinity mask: 80
```

16. Check the CPU core load to make sure the real-time task is running on the correct CPU:

```
# top  (1)
top - 10:53:04 up 21:33,  2 users,  load average: 1.88, 1.40, 1.55
Tasks: 552 total,   1 running, 551 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni, 96.6 id,  3.4 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
...
```

17. In the VM, verify correct the VM cpu loads (vCPU 1 has the real-time load in this case):

```
# top  (1)
top - 07:53:29 up 6 min,  2 users,  load average: 0.92, 0.44, 0.19
Tasks:  99 total,   1 running,  98 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  4047644 total,  2273644 free,  1642284 used,   131716 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  2249800 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  808 root      20   0  755620   3100   2840 S 100.0  0.1   2:25.87 testpmd
    1 root      20   0   38132   5408   3272 S   0.0  0.1   0:00.52 systemd
Set the OVS flows.
# ./ovs_flow_vm_vhost.sh
## cat ovs_flow_vm_vhost.sh
#! /bin/sh

# Move to command directory

cd /usr/src/ovs/utilities/

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

# Add Flow for port 0 to port 1 through VM

./ovs-ofctl add-flow br0
in_port=1,dl_type=0x800,nw_dst=5.1.1.1,idle_timeout=0,action=output:3
./ovs-ofctl add-flow br0
in_port=2,dl_type=0x800,nw_src=5.1.1.1,idle_timeout=0,action=output:4
./ovs-ofctl add-flow br0
in_port=3,dl_type=0x800,nw_src=5.1.1.1,idle_timeout=0,action=output:1
./ovs-ofctl add-flow br0
in_port=4,dl_type=0x800,nw_dst=5.1.1.1,idle_timeout=0,action=output:2
```

F.5.3.2 2 VM OVS User Space vHost Throughput Test

The 2 VM test uses 2 VMs that have the same internal setup, but affinitized to different host cores. Copying is one method to create the second VM:

1. Set up OVS for the test as described in Appendix F.2.1, OVS Base Setup for Tests.
2. Set up and run the VM for the test as described in Appendix F.5.3.1, Operating VM for (OVS User Space vHost) Throughput Test. This establishes the VM setup, DPDK, and testpmd test (or repeat for the second VM). Shut down the VM and copy it (in this example, the VM is in `/vm`, but it can be any valid directory):

```
# cd /vm
# cp Fed21-mp.qcow2 Fed21-mp_B.qcow2
```

3. Copy and change VM startup script, and edit to change the vm qcow2 filename, bridge tap, VNC output, control NIC interface MAC address, and vhost port macs and ifnames:

```
# cd /vm/vm_ctl
# cp Fed21_VM_vhost.sh Fed21_VM_vhost_B.sh
# vim Fed21_VM_vhost_B.sh
```

4. Boot the VM and set the new IP address for the VM ssh terminal control interface. The following is an example of a script to start the second VM:

```
# cat Fed21_VM_vhost_B.sh
#!/bin/sh

vm=/vm/Fed21-mp_B.qcow2
vm_name=Fed21_VM_vhost_B
vnc=15
n1=tap56
bra=br-mgt
dn_scrp_a=/vm/vm_ctl/br-mgt-ifdown
mac1=00:00:14:42:04:30

if [ ! -f $vm ];
then
    echo "VM $vm not found!"
else
    echo "VM $vm started! VNC: $vnc, net0: $n1, net1: $n2"
    tunctl -t $n1
    brctl addif $bra $n1
    ifconfig $n1 0.0.0.0 up
    taskset 38 qemu-system-x86_64 \
-m 3072 -smp 3 -cpu host -hda $vm -boot c -enable-kvm \
-pidfile /tmp/vm2.pid -monitor unix:/tmp/vm2monitor,server,nowait \
-object memory-backend-file,id=mem,size=3072M,mem-path=/dev/hugepages,share=on
\
-numa node,memdev=mem -mem-prealloc \
-net nic,model=virtio,netdev=eth0,macaddr=$mac1 \
-netdev tap,ifname=$n1,id=eth0,script=no,downscript=$dn_scrp_a \
-netdev type=tap,id=net1,script=no,downscript=no,ifname=dpdkvhost2,vhost=on \
-device virtio-net-pci,netdev=net1,mac=00:00:00:00:00:03,csum=off,gso=off,\
guest_tso4=off,guest_tso6=off,guest_ecn=off \
-netdev type=tap,id=net2,script=no,downscript=no,ifname=dpdkvhost3,vhost=on \
```

```
-device virtio-net-pci,netdev=net2,mac=00:00:00:00:00:04,csum=off,gso=off,\
guest_tso4=off,guest_tso6=off,guest_ecn=off \
-vnc :$vnc -name $vm_name &
Fi
```

Start the VM:

```
# ./Fed21_VM_vhost_B.sh
VM /vm/Fed21-mp_B.qcow2 started! VNC: 15, net0: tap56, net1:
Set 'tap56' persistent and owned by uid 0
```

5. Initialize and start the test-pmd, skipping the test-pmd build, in same way as in Appendix F.5.3.1, Operating VM for (OVS User Space vHost) Throughput Test.
6. Do the QEMU host affinitization in the same way as the first VM, except affinitize the VM vCPU1 and vCPU2 to the other available cores. For example, use core 8 (mask 100) and 9 (mask 200).
7. Start the VM 1 and affinitize as described in Operating VM for (OVS User Space vHost) Throughput Test.
8. Verify that the OVS PMD load (core 1), VM-1 testpmd load (core 6), and VM-2 testpmd load (core 8) are running on the expected host cores:

```
# top (1)
top - 10:48:22 up  1:08,  1 user,  load average: 2.85, 2.58, 2.15
Tasks: 546 total,   1 running, 545 sleeping,   0 stopped,   0 zombie
%Cpu0  :  1.4 us,  1.4 sy,  0.0 ni, 97.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
...
```

F.5.3.3 VM Series Test

The VM series test moves data bidirectionally from port input through both VMs and out the other port.

The following script sets the OVS flows for the series test:

```
# cat ovs_flow_2vm_vhost_series.sh
#! /bin/sh

# Move to command directory

cd /usr/src/ovs/utilities/

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

# Add Flows for Port 1 -> VM1 Port 3(->4) -> VM1 Port 5(->6) -> Port2
./ovs-ofctl add-flow br0
in_port=1,dl_type=0x800,nw_dst=5.1.1.1,idle_timeout=0,action=output:3
./ovs-ofctl add-flow br0
in_port=4,dl_type=0x800,nw_dst=5.1.1.1,idle_timeout=0,action=output:5
./ovs-ofctl add-flow br0
in_port=6,dl_type=0x800,nw_dst=5.1.1.1,idle_timeout=0,action=output:2

# Add Flow for Port 2 -> VM2 Port 6(->5) -> VM1 Port 4(->3) -> Port1
./ovs-ofctl add-flow br0
in_port=2,dl_type=0x800,nw_src=5.1.1.1,idle_timeout=0,action=output:6
./ovs-ofctl add-flow br0
in_port=5,dl_type=0x800,nw_src=5.1.1.1,idle_timeout=0,action=output:4
./ovs-ofctl add-flow br0
in_port=3,dl_type=0x800,nw_src=5.1.1.1,idle_timeout=0,action=output:1
```

1. Execute the script to set the flows:

   ```
   # ./ovs_flow_2vm_vhost_series.sh
   ```

2. Run the throughput test.

F.5.3.4 VM Parallel Test

The VM parallel test moves data input from first port through first VM and out the second port. The second port input data is put through the second VM and out the first port. The following script sets the parallel OVS flows for the test:

```
# cat ovs_flow_2vm_vhost_paral.sh
#! /bin/sh

# Move to command directory

cd /usr/src/ovs/utilities/

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

```
# Add Flows for Port 1->VM1 Port 3(->4)->Port 2 and Port 1->VM2 Port 6(->5)-
>Port 1
./ovs-ofctl add-flow br0
in_port=1,dl_type=0x800,nw_dst=5.1.1.1,idle_timeout=0,action=output:3
./ovs-ofctl add-flow br0
in_port=4,dl_type=0x800,nw_dst=5.1.1.1,idle_timeout=0,action=output:2

./ovs-ofctl add-flow br0
in_port=2,dl_type=0x800,nw_src=5.1.1.1,idle_timeout=0,action=output:6
./ovs-ofctl add-flow br0
in_port=5,dl_type=0x800,nw_src=5.1.1.1,idle_timeout=0,action=output:1
```

1. Execute the script to set the flows:

   ```
   # ./ovs_flow_2vm_vhost_paral.sh
   ```

2. Run the throughput test.

Appendix G SRIOV VM DPDK Test

G.1 Niantic SR-IOV VM Test

The SR-IOV test gives Niantic 10 GbE SR-IOV virtual function (VF) interfaces to a VM to do a network throughput test of the VM using SR-IOV network interface:

1. Make sure the host has boot line configuration for 1 GB hugepage memory, CPU isolation, and the Intel IOMMU is on and running in PT mode.

```
# dmesg | grep command
[    0.000000] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-3.18.5-
201.fc21.x86_64 root=UUID=cc17be46-707c-414a-8c16-11ecda5905be ro
default_hugepagesz=1GB hugepagesz=1GB hugepages=12 hugepagesz=2M hugepages=2048
isolcpus=1,2,3,4,5,6,7,28,29,30,31,32,33,34 iommu=pt intel_iommu=on rhgb quiet
```

2. The hugepage file systems are initialized.

```
# mount -t hugetlbfs nodev /dev/hugepages
# mkdir /dev/hugepages_2mb
# mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
```

3. Black list the IXGBE VF device driver to prevent the host driver being loaded when changing NIC VF configuration by adding it to */etc/modprobe.d/blacklist.conf*:

```
cat /etc/modprobe.d/blacklist.conf
. . .
# Intel ixgbe sr-iov vf (virtual driver)
blacklist ixgbevf
```

4. Uninstall and reinstall the IXGBE 10 GbE Niantic NIC device driver to enable SR_IOV and create the SR-IOV interfaces. If the control network is using the ixgbe driver, this will need to be done using the system console.

```
# modprobe -r ixgbe
# modprobe ixgbe max_vfs=1
# service network restart
```

   Note: In this case, only one SR-IOV VF was created per physical function (PF) using max_vfs=1, but often more than 1 is desired. The network needs to be restarted afterward.

5. Bring up and configure the Niantic PF interfaces to allow the SR-IOV interfaces to be used. Leaving the PF unconfigured will cause the SR-IOV interfaces to not be usable for the test. The Niantic 10 GbE interfaces happen to have default labeling of **p786p1** and **p786p2** in this example:

```
# ifconfig p786p1 11.1.1.225/24 up
# ifconfig p786p2 12.1.1.225/24 up
```

6. Find the target SR-IOV interfaces to use:

```
# lspci -nn |grep Ethernet
03:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-
Gigabit  X540-AT2 [8086:1528] (rev 01)
03:00.1 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-
Gigabit  X540-AT2 [8086:1528] (rev 01)
03:10.0 Ethernet controller [0200]: Intel Corporation X540 Ethernet Controller
Virtual  Function [8086:1515] (rev 01)
03:10.1 Ethernet controller [0200]: Intel Corporation X540 Ethernet Controller
Virtual  Function [8086:1515] (rev 01)
06:00.0 Ethernet controller [0200]: Intel Corporation 82599ES 10-Gigabit
SFI/SFP+ Network  Connection [8086:10fb] (rev 01)
06:00.1 Ethernet controller [0200]: Intel Corporation 82599ES 10-Gigabit
SFI/SFP+ Network  Connection [8086:10fb] (rev 01)
06:10.0 Ethernet controller [0200]: Intel Corporation 82599 Ethernet Controller
Virtual  Function [8086:10ed] (rev 01)
06:10.1 Ethernet controller [0200]: Intel Corporation 82599 Ethernet Controller
Virtual  Function [8086:10ed] (rev 01)
```

   We see the two Niantic (device 82599) VFs, 1 for each physical device, are PCI devices 06:10.0 and 06:10.1 with PCI device types [8086:10ed]. The numbers are dependent on the PCIe slots and, therefore, probably different for your system. Adjust the PCI device numbers for commands following your particular system's PCI device numbers.

7. Make sure the pci-stub device driver is loaded because it is used to transfer physical devices to VMs. As already noted, the pci-stub loaded by default on Fedora 21, but not on Ubuntu 14.04.

   The following code checks to determine if pci-stub is present:

```
#ls /sys/bus/pci/drivers |grep pci-stub
pci-stub
```

   If it is not present, load:

```
#modprobe pci-stub
```

8. The Niantic (82599) VF interfaces are the target interfaces which have a type ID of 8086:10ed. Load the VF devices into pci-stub.

```
# echo "8086 10ed" > /sys/bus/pci/drivers/pci-stub/new_id
```

9. List the devices owned by the pci-stub driver:

```
# ls /sys/bus/pci/drivers/pci-stub
0000:06:10.0  0000:06:10.1  bind  new_id  remove_id  uevent  unbind
```

10. If the target VF devices are not listed, they are probably attached to an IXGBEVF driver on the host. In that case, the following might be used:

```
# echo 0000:06:10.0 > /sys/bus/pci/devices/0000:06:10.0/driver/unbind
# echo 0000:06:10.1 > /sys/bus/pci/devices/0000:06:10.1/driver/unbind
# echo "8086 10ed" > /sys/bus/pci/drivers/pci-stub/new_id
```

11. Verify devices present for transfer to VM:

```
# ls /sys/bus/pci/drivers/pci-stub
0000:06:10.0  0000:06:10.1  bind  new_id  remove_id  uevent  unbind
```

G.1.1 Starting up the VM with SR-IOV

The VM is started with the SR-IOV physical devices passed to the VM in the QEMU command line. The VM is started with 3 GB of preloaded 1 GB huge memory (minimum 3 GB for one 1 GB page to be available in VM) from /dev/hugepages using 3 vCPUs of host type with first network interface device being virtio used for the bridged management network and second network device being SR-IOV 06:10.0 and third being SR-IOV 06:10.1.

Task affinitization is used to make sure execution is done using NUMA Node 0 CPU cores. If memory is available, memory is allocated on same NUMA Nodes as process is executing. The following is a script that was used to start the VM for the test affinitizing to cores 0x0e, assuming no other realtime tasks interfering (OVS with DPDK is not running):

```
# cat Fed21_VM_sriov_Niantic.sh
#!/bin/sh

vm=/vm/Fed21-mp.qcow2
vm_name=Fed21_VM_PCI
vnc=14
n1=tap55
bra=br-mgt
dn_scrp_a=/vm/vm_ctl/br-mgt-ifdown
mac1=00:00:14:42:04:29

if [ ! -f $vm ];
then
    echo "VM $vm not found!"
else
    echo "VM $vm started! VNC: $vnc, net0: $n1, net1: $n2"
    tunctl -t $n1
    brctl addif $bra $n1
    ifconfig $n1 0.0.0.0 up
    taskset 0e qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda $vm -boot c \
-pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait \
-enable-kvm -mem-path /dev/hugepages -mem-prealloc \
-net nic,model=virtio,netdev=eth0,macaddr=$mac1 \
-netdev tap,ifname=$n1,id=eth0,script=no,downscript=$dn_scrp_a \
-device pci-assign,host=06:10.0 -device pci-assign,host=06:10.1 \
-vnc :$vnc -name $vm_name &
```

Script for VM shutdown support:

```
# cat br-mgt-ifdown
#!/bin/sh
bridge='br-mgt'
/sbin/ifconfig $1 0.0.0.0 down brctl delif   ${bridge} $1
```

Start the VM:

```
# ./Fed21_VM_sriov_Niantic.sh
VM /vm/Fed21-mp.qcow2 started! VNC: 14, net0: tap55, net1:
Set 'tap55' persistent and owned by uid 0
```

G.1.2 Operating the VM for the SR-IOV Test

In the VM, the first network is configured for management network access and used for test control.

1. Verify that the kernel has bootline parameters for 1 GB hugepage support with one 1 GB page and with 2 MB hugepage support that vCPU 1 and vCPU 2 have been isolated from the VM task scheduler:

```
# dmesg |grep command
[    0.000000] Kernel command line: BOOT_IMAGE=/vmlinuz-3.18.7-200.fc21.x86_64
root=/dev/mapper/fedora--server-root ro default_hugepagesz=1GB hugepagesz=1GB
hugepages=1 hugepagesz=2M hugepages=256 isolcpus=1,2 rd.lvm.lv=fedora-
server/root rd.lvm.lv=fedora-server/swap rhgb quiet
```

   If not, configure generate new grub file and reboot VM.

```
# vim /etc/default/grub    // edit file
. . .
GRUB_CMDLINE_LINUX=" default_hugepagesz=1GB hugepagesz=1GB hugepages=1
hugepagesz=2M hugepages=256 isolcpus=1,2..."
. . .

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

# reboot
```

2. Verify 1 GB of memory page is available:

```
# cat /proc/meminfo
...
HugePages_Total:    1
HugePages_Free:     1
HugePages_Rsvd:     0
HugePages_Surp:     0
Hugepagesize:1048576 kB
DirectMap4k:  45048 kB
DirectMap2M:  2052096 kB
DirectMap1G:  1048576 kB
```

3. Check the CPU is host type (E5-2697v3 in this case) and 3 vCPUs are available:

```
# cat /proc/cpuinfo
...
processor     : 2
vendor_id     : GenuineIntel
cpu family    : 6
model  : 63
model name    : Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz stepping  : 2
microcode     : 0x1
cpu MHz       : 2593.992
cache size    : 4096 KB
physical id   : 2
siblings      : 1
core id       : 0
cpu cores     : 1
apicid : 2
```

```
initial apicid : 2 fpu     : yes
fpu_exception : yes cpuid level   : 13
wp     : yes
flags  : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc
arch_perfmon rep_good nopl eagerfpu pni pclmulqdq ssse3 fma cx16 pcid sse4_1
sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand
hypervisor lahf_lm abm xsaveopt fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
invpcid
bogomips      : 5187.98
clflush size  : 64 cache_alignment : 64
address sizes : 40 bits physical, 48 bits virtual power management:
```

4. Install DPDK:

```
# cd /usr/src/
# git clone git://dpdk.org/dpdk
# cd /usr/src/dpdk
# git checkout -b test_v1.8.0 v1.8.0
# make install T=x86_64-ivshmem-linuxapp-gcc
```

5. Move to the l3fwd-vf directory and increase the NIC queue size and increase buffer count with the editor, then verify with git (if using git):

```
# cd /usr/src/dpdk/examples/l3fwd-vf
# git diff
diff --git a/examples/l3fwd-vf/main.c b/examples/l3fwd-vf/main.c
index 7cc7228..0066ce0 100644
--- a/examples/l3fwd-vf/main.c
+++ b/examples/l3fwd-vf/main.c
@@ -108,7 +108,7 @@
                                nb_ports*nb_lcores*MAX_PKT_BURST +
                        \
                                nb_ports*n_tx_queue*RTE_TEST_TX_DESC_DEFAULT +
        \
                                nb_lcores*MEMPOOL_CACHE_SIZE),
                        \
-                               (unsigned)8192)
+                               (unsigned)16384)

 /*
  * RX and TX Prefetch, Host, and Write-back threshold values should be
@@ -142,8 +142,8 @@
 /*
  * Configurable number of RX/TX ring descriptors
  */
-#define RTE_TEST_RX_DESC_DEFAULT 128
-#define RTE_TEST_TX_DESC_DEFAULT 512
+#define RTE_TEST_RX_DESC_DEFAULT 2048
+#define RTE_TEST_TX_DESC_DEFAULT 2048
 static uint16_t nb_rxd = RTE_TEST_RX_DESC_DEFAULT;
 static uint16_t nb_txd = RTE_TEST_TX_DESC_DEFAULT;
```

6. Build the l3fwd-vf example program:

```
# export RTE_SDK=/usr/src/dpdk
# export RTE_TARGET=x86_64-ivshmem-linuxapp-gcc
# make
  CC main.o
  LD l3fwd-vf
  INSTALL-APP l3fwd-vf
  INSTALL-MAP l3fwd-vf.map
```

7. Mount the hugepage file system:

```
# mount -t hugetlbfs nodev /dev/hugepage
# mkdir /dev/hugepages_2mb
# mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
```

8. Install the `UIO` kernel drivers in the VM:

```
# modprobe uio
# insmod /usr/src/dpdk/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko
```

9. Find the SR-IOV devices:

```
# lspci -nn
00:00.0 Host bridge [0600]: Intel Corporation 440FX - 82441FX PMC [Natoma]
[8086:1237] (rev 02)
00:01.0 ISA bridge [0601]: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton
II] [8086:7000]
00:01.1 IDE interface [0101]: Intel Corporation 82371SB PIIX3 IDE
[Natoma/Triton II] [8086:7010]
00:01.3 Bridge [0680]: Intel Corporation 82371AB/EB/MB PIIX4 ACPI [8086:7113]
(rev 03)
00:02.0 VGA compatible controller [0300]: Cirrus Logic GD 5446 [1013:00b8]
00:03.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
00:04.0 Ethernet controller [0200]: Intel Corporation XL710/X710 Virtual
Function [8086:154c] (rev 01)
00:05.0 Ethernet controller [0200]: Intel Corporation XL710/X710 Virtual
Function [8086:154c] (rev 01)
```

The networks are in the order of the QEMU command line. The first network interface, 00:03.0 is the bridged virtio control network. The second network interface 00:04.0 is the first SR-IOV physical PCI device passed on the QEMU command line with the third network interface 00:05.0 is the second SR-IOV physical PCI device passed.

10. Bind the SR-IOV devices to the igb_uio device driver:

```
# /usr/src/dpdk/tools/dpdk_nic_bind.py --bind=igb_uio 00:04.0
# /usr/src/dpdk/tools/dpdk_nic_bind.py --bind=igb_uio 00:05.0
# /usr/src/dpdk/tools/dpdk_nic_bind.py --status
```

11. Run the l3fwd-vf example program:

```
# cd /usr/src/dpdk/examples/l3fwd-vf
# ./build/l3fwd-vf -c 6 -n 4 --socket-mem 1024 -- -p 0x3 --
config="(0,0,1),(1,0,2)" &
```

The -c 6 and -config parameters correctly affinitizes the l3fwd-vf task to the correct vCPUs within the VM. Because the CPU cores on the host are isolated and QEMU task was started on isolated cores, however, the real-time l3fwd-vf task may be thrashing since both vCPUs are probably running on the same CPU core, requiring the CPU affinitization to set correctly on the host, or poor network performance will occur.

12. Locate the vCPU QEMU thread IDs on the host:

```
# ps -eLF|grep qemu
root       2279      1   2279  0    5 974255 44080  1 10:13 pts/0    00:00:02
/usr/bin/qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -
boot c -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
enable-kvm -mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:10.0 -device pci-assign,host=06:10.1 -vnc :14 -name
Fed21_VM_PCI
root       2279      1   2281  1    5 974255 44080  1 10:13 pts/0    00:00:31
/usr/bin/qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -
boot c -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
enable-kvm -mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:10.0 -device pci-assign,host=06:10.1 -vnc :14 -name
Fed21_VM_PCI
root       2279      1   2282 36    5 974255 44080  1 10:13 pts/0    00:13:39
/usr/bin/qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -
boot c -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
enable-kvm -mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:10.0 -device pci-assign,host=06:10.1 -vnc :14 -name
Fed21_VM_PCI
root       2279      1   2283 36    5 974255 44080  1 10:13 pts/0    00:13:39
/usr/bin/qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -
boot c -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
enable-kvm -mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:10.0 -device pci-assign,host=06:10.1 -vnc :14 -name
Fed21_VM_PCI
root       2279      1   2285  0    5 974255 44080  1 10:13 pts/0    00:00:00
/usr/bin/qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -
boot c -pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -
enable-kvm -mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:10.0 -device pci-assign,host=06:10.1 -vnc :14 -name
Fed21_VM_PCI
```

We see that QEMU thread task IDs 2282 and 2283 are accumulating a process time of 13:39 and 13:39, respectively, compared to the other low-usage threads in this case. We know that the two 100% load DPDK treads are running on vCPU1 and vCPU2, respectively. This indicates that vCPU1 is task ID 2282 and vCPU2 is task ID 2283. Because QEMU creates vCPU task threads sequentially, vCPU0 must be QEMU thread task 2281.

13. Having CPU core 4 and 5 available on this 10 CPU core processors, we will move vCPU1 to CPU core 4 and vCPU2 to CPU core 5 for the test:

```
# taskset -p 10 2282
pid 2282's current affinity mask: e
pid 2282's new affinity mask: 10
# taskset -p 20 2283
pid 2283's current affinity mask: e
pid 2283's new affinity mask: 20
```

14. Using top, we verify that tasks are running on target CPU cores:

```
# top  (1)
top - 10:51:34 up 48 min,  3 users,  load average: 1.95, 1.97, 1.71
Tasks: 556 total,   1 running, 555 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.6 sy,  0.0 ni, 99.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu24 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu25 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu26 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu27 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu28 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu29 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu30 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu31 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu32 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu33 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu34 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu35 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu36 :  0.0 us,  0.6 sy,  0.0 ni, 99.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu37 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

```
%Cpu38 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu39 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu40 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu41 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu42 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu43 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu44 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu45 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu46 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu47 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu48 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu49 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu50 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu51 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu52 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu53 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu54 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu55 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 65907196 total, 48153644 free, 17319520 used,   434032 buff/cache
KiB Swap:  8388604 total,  8388604 free,        0 used. 48255972 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  2279 root      20   0 3897020  44080  21516 S 200.6  0.1  29:24.99 qemu-
system-x86
  2337 root      20   0  147012   4416   3264 R   1.2  0.0   0:00.03 top
     1 root      20   0   45028  10800   3408 S   0.0  0.0   0:03.88 systemd
```

This shows that the DPDK l3fwd CPU 100% loads are on the target CPU cores 4 and 5.

15. Check for the correct vCPU loads in the VM:

```
# top  (1)

top - 07:52:14 up 39 min,  1 user,  load average: 2.00, 1.99, 1.70
Tasks:  96 total,   2 running,  94 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3081508 total,  1306092 free,  1634400 used,   141016 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1317048 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  977 root      20   0 1673872   3152   2892 R 199.7  0.1  30:06.23 l3fwd-vf
    1 root      20   0   38132   5492   3340 S   0.0  0.2   0:00.64 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.39 ksoftirqd/0
```

In the VM, the top utility shows that the DPDK l3task-vf loads are on vCPU1 and vCPU2. Before correct host affiliation due to CPU isolation, the vCPU loads were running on the same host core because the CPU load scheduling is disabled, causing the top utility not to show close to 100% user task loads in the VM. The VM is now ready for the SR-IOV throughput test.

In a production system, the VM startup and affiliation would be done in a different manner to avoid disrupting any real-time processes currently running on the host or other VMs.

Appendix H VM PCI Passthrough

H.1 Niantic PCI Passthrough VM Test

H.1.1 Starting up the Niantic PCI Passthrough Host

The PCI passthrough test gives Niantic 10 GbE PF interfaces to a VM to do a network throughput test using the physical network interfaces owned by the VM:

1. Make sure the host has a boot-line configuration for 1 GB of hugepage memory, CPU isolation, and the Intel IOMMU is on and running in PT mode:

```
# dmesg | grep command
[    0.000000] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-3.18.5-201.fc21.x86_64 root=UUID=cc17be46-707c-414a-8c16-11ecda5905be ro default_hugepagesz=1GB hugepagesz=1GB hugepages=12 hugepagesz=2M hugepages=2048 isolcpus=1,2,3,4,5,6,7,28,29,30,31,32,33,34 iommu=pt intel_iommu=on rhgb quiet
```

2. The hugepage file systems are initialized:

```
# mount -t hugetlbfs nodev /dev/hugepages
# mkdir /dev/hugepages_2mb
# mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
```

3. Find the Ethernet interfaces. In this case, 06:00:1 and 06:00.1 are the target interfaces, if different PCI IDs, have to readjust commands to your PCI IDs.

```
# lspci -nn |grep Ethernet
03:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-Gigabit X540-AT2 [8086:1528] (rev 01)
03:00.1 Ethernet controller [0200]: Intel Corporation Ethernet Controller 10-Gigabit X540-AT2 [8086:1528] (rev 01)
06:00.0 Ethernet controller [0200]: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection [8086:10fb] (rev 01)
06:00.1 Ethernet controller [0200]: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection [8086:10fb] (rev 01)
```

4. Make sure the target Ethernet interfaces are down (eth3 and eth4 are arbitrary names; use your system's interface names:

```
# ifconfig eth3 down
# ifconfig eth4 down
```

5. Unbind target physical Ethernet interfaces from the current device driver (ixgbe driver):

```
# echo 0000:06:00.0 > /sys/bus/pci/devices/0000:06:00.0/driver/unbind
# echo 0000:06:00.1 > /sys/bus/pci/devices/0000:06:00.1/driver/unbind
```

6. Bind to the PCI stub driver. You need to use the PCI device type id, as displayed by `lspci -nn` command, which in this case is [8086:10fb]. Bound devices will not attach.

```
# echo -n "8086 10fb" > /sys/bus/pci/drivers/pci-stub/new_id
```

7. Check that the target devices are attached to the pci-stub driver. In this case 06:00.0 and 06:00.1 are shown in the directory list:

```
#  ls /sys/bus/pci/drivers/pci-stub
0000:06:00.0  0000:06:00.1  bind  new_id  remove_id  uevent  unbind
```

H.1.2 Starting up the Niantic PCI Passthrough VM

The VM is started with the PF devices passed to the VM in the QEMU command line. The VM is started with 3 GB of preloaded 1 GB huge memory (a minimum 3 GB for one 1 GB page to be available in VM) from */dev/hugepages,* using 3 vCPUs of host type. The first network interface device (virtio) is used for the bridged management network, the second is the Niantic PF PCI device 06:10.0, and the third is the Niantic PF PCI device 06:10.1.

Task affinitization is used to ensure execution is done using NUMA node 0 CPU cores. If memory is available, it is allocated on the same NUMA nodes as the process is executing. The following is a script that used to start the VM for the test affinitizing to cores 0x0e. It assumes no other real-time tasks are interfering (OVS with DPDK is not running):

```
# cat Fed21_VM_pci.sh
#!/bin/sh

vm=/vm/Fed21-mp.qcow2
vm_name=Fed21_VM_PCI
vnc=14
n1=tap55
bra=br-mgt
dn_scrp_a=/vm/vm_ctl/br-mgt-ifdown
mac1=00:00:14:42:04:29

if [ ! -f $vm ];
then
    echo "VM $vm not found!"
else
    echo "VM $vm started! VNC: $vnc, net0: $n1, net1: $n2"
    tunctl -t $n1
    brctl addif $bra $n1
    ifconfig $n1 0.0.0.0 up
    taskset 0e qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda $vm -boot c \
-pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait \
-enable-kvm -mem-path /dev/hugepages -mem-prealloc \
-net nic,model=virtio,netdev=eth0,macaddr=$mac1 \
-netdev tap,ifname=$n1,id=eth0,script=no,downscript=$dn_scrp_a \
-device pci-assign,host=06:00.0 \
-device pci-assign,host=06:00.1 \
-vnc :$vnc -name $vm_name &
Fi
```

Script for VM shutdown support:

```
# cat br-mgt-ifdown
#!/bin/sh
bridge='br-mgt'
/sbin/ifconfig $1 0.0.0.0 down brctl delif   ${bridge} $1
```

Start the VM:

```
# ./Fed21_VM_pci.sh
VM /vm/Fed21-mp.qcow2 started! VNC: 14, net0: tap55, net1:
Set 'tap55' persistent and owned by uid 0
```

H.1.3 Setting up the Niantic PCI Passthrough VM

In the VM, the first network is configured for management network access and used for test control.

1. Verify that the kernel has boot-line parameters for 1 GB hugepage support with one 1 GB page and 2 MB hugepage support and that vCPU 1 and vCPU 2 have been isolated from the VM task scheduler:

   ```
   # dmesg |grep command
   [    0.000000] Kernel command line: BOOT_IMAGE=/vmlinuz-3.18.7-200.fc21.x86_64
   root=/dev/mapper/fedora--server-root ro default_hugepagesz=1GB hugepagesz=1GB
   hugepages=1 hugepagesz=2M hugepages=256 isolcpus=1,2 rd.lvm.lv=fedora-
   server/root rd.lvm.lv=fedora-server/swap rhgb quiet
   ```

   If not, configure generate new grub file and reboot VM.

   ```
   # vim /etc/default/grub    // edit file
   . . .
   GRUB_CMDLINE_LINUX=" default_hugepagesz=1GB hugepagesz=1GB hugepages=1
   hugepagesz=2M hugepages=256 isolcpus=1,2..."
   . . .

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

   # reboot
   ```

2. Verify 1 GB memory page is available:

   ```
   # cat /proc/meminfo
   ...
   HugePages_Total:    1
   HugePages_Free:     1
   HugePages_Rsvd:     0
   HugePages_Surp:     0
   Hugepagesize:1048576 kB
   DirectMap4k:  45048 kB
   DirectMap2M:  2052096 kB
   DirectMap1G:  1048576 kB
   ```

3. Check the CPU is host type (E5-2697v3 in this case) and 3 vCPUs are available:

```
# cat /proc/cpuinfo
...
processor     : 2
vendor_id     : GenuineIntel
cpu family    : 6
model  : 63
model name    : Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz stepping  : 2
microcode     : 0x1
cpu MHz       : 2593.992
cache size    : 4096 KB
physical id   : 2
siblings      : 1
core id       : 0
cpu cores     : 1
apicid : 2
initial apicid : 2 fpu     : yes
fpu_exception : yes cpuid level   : 13
wp     : yes
flags  : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc
arch_perfmon rep_good nopl eagerfpu pni pclmulqdq ssse3 fma cx16 pcid sse4_1
sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand
hypervisor lahf_lm abm xsaveopt fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
invpcid
bogomips      : 5187.98
clflush size  : 64 cache_alignment : 64
address sizes : 40 bits physical, 48 bits virtual power management:
```

4. Install DPDK:

```
# cd /usr/src/
# git clone git://dpdk.org/dpdk
# cd /usr/src/dpdk
# git checkout -b test_v1.8.0 v1.8.0
# make install T=x86_64-ivshmem-linuxapp-gcc
```

5. Mount the hugepage file system:

```
# mount -t hugetlbfs nodev /dev/hugepage
# mkdir /dev/hugepages_2mb
# mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
```

6. Install `UIO` kernel drivers in VM:

```
# modprobe uio
# insmod /usr/src/dpdk/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko
```

7. Find PCI passthrough devices in VM:

```
# lspci -nn
00:00.0 Host bridge [0600]: Intel Corporation 440FX - 82441FX PMC [Natoma]
[8086:1237] (rev 02)
00:01.0 ISA bridge [0601]: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton
II] [8086:7000]
00:01.1 IDE interface [0101]: Intel Corporation 82371SB PIIX3 IDE
[Natoma/Triton II] [8086:7010]
```

```
00:01.3 Bridge [0680]: Intel Corporation 82371AB/EB/MB PIIX4 ACPI [8086:7113]
(rev 03)
00:02.0 VGA compatible controller [0300]: Cirrus Logic GD 5446 [1013:00b8]
00:03.0 Ethernet controller [0200]: Red Hat, Inc Virtio network device
[1af4:1000]
00:04.0 Ethernet controller [0200]: Intel Corporation 82599ES 10-Gigabit
SFI/SFP+ Network Connection [8086:10fb] (rev 01)
00:05.0 Ethernet controller [0200]: Intel Corporation 82599ES 10-Gigabit
SFI/SFP+ Network Connection [8086:10fb] (rev 01)
```

The networks are in the order of the QEMU command line. The first network interface, 00:03.0 is the bridged virtio control network. The second network interface 00:04.0 is the first PCI passthrough physical PCI device passed on the QEMU command line with the third network interface 00:05.0 is the second PCI passthrough physical PCI device passed.

8. Bind the Niantic PF devices to the igb_uio device driver.

```
# /usr/src/dpdk/tools/dpdk_nic_bind.py --bind=igb_uio 00:04.0
# /usr/src/dpdk/tools/dpdk_nic_bind.py --bind=igb_uio 00:05.0
# /usr/src/dpdk/tools/dpdk_nic_bind.py --status
```

H.1.4 VM l3fwd Niantic PCI Passthrough Test

1. Initialize VM as described in section H.1.2 Niantic PCI Passthrough VM Startup.
2. Move to the l3fwd directory and increase NIC queue size and increase buffer count with editor and verify with git (if using git):

```
# cd /usr/src/dpdk/examples/l3fwd
# vim main.c
# git diff -M -C

diff --git a/examples/l3fwd/main.c b/examples/l3fwd/main.c
index 918f2cb..a1c8a9a 100644
--- a/examples/l3fwd/main.c
+++ b/examples/l3fwd/main.c
@@ -133,7 +133,7 @@
                                nb_ports*nb_lcores*MAX_PKT_BURST +
                                \
                                nb_ports*n_tx_queue*RTE_TEST_TX_DESC_DEFAULT +
                 \
                                nb_lcores*MEMPOOL_CACHE_SIZE),
                                \
-                               (unsigned)8192)
+                               (unsigned)16384)

 #define MAX_PKT_BURST     32
 #define BURST_TX_DRAIN_US 100 /* TX drain every ~100us */
@@ -156,8 +156,8 @@
 /*
  * Configurable number of RX/TX ring descriptors
  */
-#define RTE_TEST_RX_DESC_DEFAULT 128
-#define RTE_TEST_TX_DESC_DEFAULT 512
```

```
+#define RTE_TEST_RX_DESC_DEFAULT 2048
+#define RTE_TEST_TX_DESC_DEFAULT 2048
 static uint16_t nb_rxd = RTE_TEST_RX_DESC_DEFAULT;
 static uint16_t nb_txd = RTE_TEST_TX_DESC_DEFAULT;
```

3. Build l3fwd example program:

```
# export RTE_SDK=/usr/src/dpdk
# export RTE_TARGET=x86_64-ivshmem-linuxapp-gcc
# make
  CC main.o
  LD l3fwd
  INSTALL-APP l3fwd
  INSTALL-MAP l3fwd.map
```

4. Run the l3fwd example program:

```
# cd /usr/src/dpdk/examples/l3fwd
# ./build/l3fwd -c 6 -n 4 --socket-mem 1024 -- -p 0x3 --
config="(0,0,1),(1,0,2)" &
```

The -c 6 and -config parameters correctly affinitizes the l3fwd task to the correct vCPUs within the VM; however, since the CPU cores on the host are isolated and QEMU task was started on isolated cores, the real-time l3fwd task could be thrashing because both vCPUs are probably running on the same CPU core. This requires the CPU affinitization to set correctly on the host, or poor zero loss network performance will occur.

5. Locate the vCPU QEMU thread IDs on the host:

```
# ps -eLF| grep qemu
root       1589      1   1589  0    5 974515 44300  1 15:34 pts/0    00:00:01
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
Fed21_VM_PCI
root       1589      1   1591  2    5 974515 44300  1 15:34 pts/0    00:00:19
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
Fed21_VM_PCI
root       1589      1   1592  4    5 974515 44300  1 15:34 pts/0    00:00:29
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
Fed21_VM_PCI
root       1589      1   1593  4    5 974515 44300  1 15:34 pts/0    00:00:28
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
```

```
Fed21_VM_PCI
root       1589      1   1597  0    5 974515 44300  1 15:34 pts/0    00:00:00
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
Fed21_VM_PCI
```

We see that QEMU thread task IDs 1592 and 1593 are accumulating process time of 0:29 and 0:28, respectively, compared to the other low usage threads in this case. We know that the two 100% load DPDK treads are running on vCPU1 and vCPU2, respectively; so this indicates that vCPU1 is task ID 1592 and vCPU2 is task ID 1593. Because QEMU creates vCPU task threads sequentially, vCPU0 must be QEMU thread task 1591.

6. Having CPU core 6 and 7 available on this 10 CPU core processors, we move vCPU1 to CPU core 6 and vCPU2 to CPU core 7 for the test:

```
# taskset -p 40 1592
pid 1592's current affinity mask: e
pid 1592's new affinity mask: 40
# taskset -p 80 1593
pid 1593's current affinity mask: e
pid 1593's new affinity mask: 80
```

7. Using `top`, we verify that the tasks are running on target CPU cores:

```
# top  (1)
top - 15:48:04 up 23 min,  1 user,  load average: 1.84, 0.87, 0.37
Tasks: 544 total,   1 running, 543 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.9 us,  0.0 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.0 us,  0.9 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu24 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

```
%Cpu25 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu26 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu27 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu28 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu29 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu30 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu31 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu32 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu33 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu34 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu35 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu36 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu37 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu38 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu39 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu40 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu41 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu42 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu43 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu44 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu45 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu46 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu47 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu48 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu49 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu50 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu51 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu52 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu53 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu54 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu55 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 65907200 total, 43854296 free, 21609592 used,   443312 buff/cache
KiB Swap:  8388604 total,  8388604 free,        0 used. 43978580 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  1589 root      20   0 3898060  44300  21360 S 200.0  0.1   3:56.30 qemu-
system-x86
  1668 root      20   0  146884   4260   3112 R   1.9  0.0   0:00.03 top
```

8. Check for the correct vCPU loads in the VM:

```
# top  (1)
top - 12:48:36 up 13 min,  1 user,  load average: 1.92, 0.97, 0.40
Tasks:  94 total,   2 running,  92 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  1.7 sy,  0.0 ni, 93.2 id,  5.1 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3081508 total,  1274536 free,  1632228 used,   174744 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1319440 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  918 root      20   0 1674856   3148   2888 R 200.0  0.1   4:35.31 l3fwd
    1 root      20   0   38132   5576   3432 S   0.0  0.2   0:00.64 systemd
```

```
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
```

In the VM, the top utility shows that the DPDK l3task loads are on vCPU1 and vCPU2. Before correct host affiliation, due to CPU isolation the vCPU loads were running on the same host core since CPU load scheduling is disabled causing top to not show close to 100% user task loads in the VM. The VM is now ready for the PCI passthrough throughput test.

In a production system, the VM startup and affiliation would be done in a different manner to avoid disrupting any real-time processes currently running on the host or other VMs.

H.1.5 VM l2fwd Niantic PCI Passthrough Test

1. Initialize VM as described in section H.1.2, Niantic PCI Passthrough VM Startup.
2. Move to the l2fwd directory and increase NIC Queue size to 2K and increase buffer count to 32K with editor and verify with git (if using git):

```
# cd /usr/src/dpdk/examples/l2fwd
# vim main.c
# git diff -M -C
diff --git a/examples/l2fwd/main.c b/examples/l2fwd/main.c
index e684234..3748eeb 100644
--- a/examples/l2fwd/main.c
+++ b/examples/l2fwd/main.c
@@ -73,7 +73,7 @@
 #define RTE_LOGTYPE_L2FWD RTE_LOGTYPE_USER1

 #define MBUF_SIZE (2048 + sizeof(struct rte_mbuf) + RTE_PKTMBUF_HEADROOM)
-#define NB_MBUF   8192
+#define NB_MBUF   32768

 #define MAX_PKT_BURST 32
 #define BURST_TX_DRAIN_US 100 /* TX drain every ~100us */
@@ -81,8 +81,8 @@
 /*
  * Configurable number of RX/TX ring descriptors
  */
-#define RTE_TEST_RX_DESC_DEFAULT 128
-#define RTE_TEST_TX_DESC_DEFAULT 512
+#define RTE_TEST_RX_DESC_DEFAULT 2048
+#define RTE_TEST_TX_DESC_DEFAULT 2048
 static uint16_t nb_rxd = RTE_TEST_RX_DESC_DEFAULT;
 static uint16_t nb_txd = RTE_TEST_TX_DESC_DEFAULT;
```

3. Build the l2fwd example program:

```
# export RTE_TARGET=x86_64-ivshmem-linuxapp-gcc
# export RTE_SDK=/usr/src/dpdk
# make
  CC main.o
  LD l2fwd
  INSTALL-APP l2fwd
  INSTALL-MAP l2fwd.map
```

4. Run l2fwd example program:

```
./build/l2fwd -c 6 -n 4 --socket-mem 1024,0 -- -p 3 &
```

The -c 6 and -config parameters correctly affinitizes the l3fwd task to the correct vCPUs within the VM; however, because the CPU cores on the host are isolated and QEMU task was started on isolated cores, the real-time l3fwd task could be thrashing due to both vCPUs probably running on the same CPU core. This requires the CPU affinitization to set correctly on the host, or poor zero loss network performance will occur.

5. Locate the vCPU QEMU thread IDs on the host (Same as the PCI-Passthrough l3fwd test since l2fwd was executed afterward):

```
# ps -eLF| grep qemu
root       1589      1   1589  0    5 974515 44300  1 15:34 pts/0    00:00:01
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
Fed21_VM_PCI
root       1589      1   1591  2    5 974515 44300  1 15:34 pts/0    00:00:19
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
Fed21_VM_PCI
root       1589      1   1592  4    5 974515 44300  1 15:34 pts/0    00:00:29
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
Fed21_VM_PCI
root       1589      1   1593  4    5 974515 44300  1 15:34 pts/0    00:00:28
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
Fed21_VM_PCI
root       1589      1   1597  0    5 974515 44300  1 15:34 pts/0    00:00:00
qemu-system-x86_64 -m 3072 -smp 3 -cpu host -hda /vm/Fed21-mp.qcow2 -boot c -
pidfile /tmp/vm1.pid -monitor unix:/tmp/vm1monitor,server,nowait -enable-kvm -
mem-path /dev/hugepages -mem-prealloc -net
nic,model=virtio,netdev=eth0,macaddr=00:00:14:42:04:29 -netdev
tap,ifname=tap55,id=eth0,script=no,downscript=/vm/vm_ctl/br-mgt-ifdown -device
pci-assign,host=06:00.0 -device pci-assign,host=06:00.1 -vnc :14 -name
Fed21_VM_PCI
```

We see that QEMU thread task IDs 1592 and 1593 are accumulating process time of 0:29 and 0:28, respectively, compared to the other low-usage threads in this case. We know that the two 100% load DPDK treads are running on vCPU1 and vCPU2, respectively. This indicates that vCPU1 is task ID 1592 and vCPU2 is task ID 1593. Because QEMU creates vCPU task threads sequentially, vCPU0 must be QEMU thread task 1591.

6. Having CPU core 6 and 7 available on this 10 CPU core processors, we move vCPU1 to CPU core 6 and vCPU2 to CPU core 7 for the test:

```
# taskset -p 40 1592
pid 1592's current affinity mask: e
pid 1592's new affinity mask: 40
# taskset -p 80 1593
pid 1593's current affinity mask: e
pid 1593's new affinity mask: 80
```

7. Using `top`, we verify that tasks are running on target CPU cores:

```
# top  (1)
top - 15:48:04 up 23 min,  1 user,  load average: 1.84, 0.87, 0.37
Tasks: 544 total,   1 running, 543 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.9 us,  0.0 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.0 us,  0.9 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu24 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu25 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu26 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu27 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu28 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu29 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu30 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu31 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu32 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu33 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu34 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu35 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu36 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu37 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu38 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu39 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

```
%Cpu40 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu41 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu42 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu43 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu44 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu45 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu46 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu47 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu48 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu49 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu50 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu51 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu52 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu53 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu54 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu55 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 65907200 total, 43854296 free, 21609592 used,   443312 buff/cache
KiB Swap:  8388604 total,  8388604 free,        0 used. 43978580 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  1589 root      20   0 3898060  44300  21360 S 200.0  0.1   3:56.30 qemu-
system-x86
  1668 root      20   0  146884   4260   3112 R   1.9  0.0   0:00.03 top
```

8. Check for the correct vCPU loads in the VM:

```
# top  (1)
top - 12:48:36 up 13 min,  1 user,  load average: 1.92, 0.97, 0.40
Tasks:  94 total,   2 running,  92 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  1.7 sy,  0.0 ni, 93.2 id,  5.1 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3081508 total,  1274536 free,  1632228 used,   174744 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1319440 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  918 root      20   0 1674856   3148   2888 R 200.0  0.1   4:35.31 l2fwd
    1 root      20   0   38132   5576   3432 S   0.0  0.2   0:00.64 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
```

In the VM, the top utility shows that the DPDK l3task loads are on vCPU1 and vCPU2. Before correct host affiliation, due to CPU isolation the vCPU loads were running on the same host core since CPU load scheduling is disabled causing `top` to not show close to 100% user task loads in the VM. The VM is now ready for the PCI passthrough throughput test.

In a production system, the VM startup and affiliation would be done in a different manner to avoid disrupting any real time processes currently running on the host or other VMs.

Appendix I Intel® QuickAssist

I.1 Installing Intel® QuickAssist

When setting up an Intel® QuickAssist (QAT) PCI device for passthrough to a VM, make sure that VT-d is enabled in the BIOS and "`intel_iommu=on iommu=pt`" is used in the *grub.cfg* file to boot the OS with IOMMU enabled. The VM has access to the QAT PCI device using PCI passthrough. The VM uses standard Open vSwitch virtio with the standard vhost IO virtualization method for networking.

1. Run the following command to verify that the host has two QAT devices provisioned in it:

```
# lspci -nn |grep 043
0c:00.0 Co-processor [0b40]: Intel Corporation Coleto Creek PCIe Endpoint
[8086:0435]
85:00.0 Co-processor [0b40]: Intel Corporation Coleto Creek PCIe Endpoint
[8086:0435]
```

2. Use the following commands to detach PCI devices from the host:

```
# echo 0000:85:00.0 > /sys/bus/pci/devices/0000\:85\:00.0/driver/unbind
# echo 0000:0c:00.0 > /sys/bus/pci/devices/0000\:0c\:00.0/driver/unbind
# echo 8086 0435 > /sys/bus/pci/drivers/pci-stub/new_id
```

   Note: You may need to use the specific PCI bus ID per your system setup.

3. After detaching the acceleration complex from the host operating system, bind the appropriate bus/device/function to pci-stub driver:

```
# echo 0000:85:00.0 > /sys/bus/pci/drivers/pci-stub/bind
# echo 0000:0c:00.0 > /sys/bus/pci/drivers/pci-stub/bind
```

4. Verify if the devices are bound to the pci-stub:

```
# lspci -vv |grep pci-stub
```

5. On a separate compute node that uses standard Open vSwitch for networking, add an ovs bridge called br0 (assuming br0 not already used). The tap devices tap1, tap2, tap3 and tap4 are used as data network vNICs for the two VMs. Each of the two 10 GbE on the host are bridged to the ovs bridge br0 as follows:

```
# ovs-vsctl add-br br0
# tunctl -t tap1
# tunctl -t tap2
# tunctl -t tap3
# tunctl -t tap4
# ovs-vsctl add-port br0 tap1
# ovs-vsctl add-port br0 tap2
# ovs-vsctl add-port br0 tap3
# ovs-vsctl add-port br0 tap4
# ovs-vsctl add-port br0 p786p1
# ovs-vsctl add-port br0 p786p2
```

6. Bring up the tap devices and ports added to the bridge:

```
# ip link set br0 up
# ip link set tap1 up
# ip link set tap2 up
# ip link set tap3 up
# ip link set tap4 up
# ip link set p786p1 up
# ip link set p786p2 up
```

7. Disable Linux Kernel forwarding on the host:

```
# echo 0 > /proc/sys/net/ipv4/ip_forward
```

8. For best performance the QEMU tasks need to be affinitized to the NUMA node CPUs as the QAT cards are inserted. In this case, both QAT cards are inserted into PCIe slots that are attached to CPU socket 2, which is NUMA node 1. For an 18 core CPU (used for this test), first cores of the hyperthreaded pair are cores 18 to 35 (core mask 0FFFFC0000). For the 14 core CPU, the first cores of the hyperthreaded pair are cores 14 to 27 (core mask 0FFFC000), but not shown in this doc. The memory is allocated from the NUMA node that the task is running on, so the task needs to be affinitized on startup.

9. The two VMs are started with the QAT devices give to the VM using PCI passthrough. The following is a `qemu` command to pass 85:00.0 and 0c:00.0 to the VMs on NUMA node 1:

```
# taskset 3c0000 qemu-system-x86_64 -cpu host -enable-kvm -hda <VM1_image_path>
-m 8192 -smp 4 -net nic,model=virtio,netdev=eth0,macaddr=00:00:00:00:00:01 -
netdev  tap,ifname=tap1,id=eth0,vhost=on,script=no,downscript=no -net
nic,model=virtio,netdev=eth1,macaddr=00:00:00:00:00:02 -netdev
tap,ifname=tap2,id=eth1,vhost=on,script=no,downscript=no -vnc :15 -name vm1 -
device pci- assign,host=85:00.0 &
# taskset 3c000 qemu-system-x86_64 -cpu host -enable-kvm -hda <VM2_image_path>
-m 8192 -smp 4 -net nic,model=virtio,netdev=eth0,macaddr=00:00:00:00:00:03 -
netdev  tap,ifname=tap3,id=eth0,vhost=on,script=no,downscript=no -net
nic,model=virtio,netdev=eth1,macaddr=00:00:00:00:00:04 -netdev
tap,ifname=tap4,id=eth1,vhost=on,script=no,downscript=no -vnc :16 -name vm2 -
device pci- assign,host=0c:00.0 &
```

10. The virtual CPUs (4 vCPUs each) of the two VMs need to be affinitized to the target NUMA node 1 CPU cores (0FF00000 in this case). Will need to find the associated CPU cores:

```
# ps -eLF | grep qemu
root      2449     1  2449  3    6 2320338 8402504 28 Th0324 ?    00:03:36
qemu-system-x86_64 -cpu host -enable-kvm -hda vm2-hda.qcow2 -m 8192 -smp 4 -net
nic,model=virtio,netdev=eth0,macaddr=00:00:00:00:00:03 -netdev
tap,ifname=tap3,id=eth0,vhost=on,script=no,downscript=no -net
nic,model=virtio,netdev=eth1,macaddr=00:00:00:00:00:04 -netdev
tap,ifname=tap4,id=eth1,vhost=on,script=no,downscript=no -vnc :16 -name vm2
root      2449     1  2457 15    6 2320338 8402504 28 . . .
root      2449     1  2458 13    6 2320338 8402504 28 . . .
root      2449     1  2459 12    6 2320338 8402504 28 . . .
root      2449     1  2460 12    6 2320338 8402504 28 . . .
root      2449     1  2462  0    6 2320338 8402504 28 . . .
root      3169     1  3169  1    7 2303954 817364 18 00:41 pts/0  00:00:00
qemu-system-x86_64 -cpu host -enable-kvm -hda vm1-hda.qcow2 -m 8192 -smp 4 -net
nic,model=virtio,netdev=eth0,macaddr=00:00:00:00:00:01 -netdev
tap,ifname=tap1,id=eth0,vhost=on,script=no,downscript=no -net
nic,model=virtio,netdev=eth1,macaddr=00:00:00:00:00:02 -netdev
tap,ifname=tap2,id=eth1,vhost=on,script=no,downscript=no -vnc :15 -name vm1
root      3169     1  3176 18    7 2303954 817364 18 . . .
```

```
root      3169     1  3177  6    7 2303954 817364 18 . . .
root      3169     1  3178  5    7 2303954 817364 18 . . .
root      3169     1  3179  4    7 2303954 817364 18 . . .
root      3169     1  3181  0    7 2303954 817364 18 . . .
root      3169     1  3188  0    7 2303954 817364 18 . . .
```

In this case, the VM1 virtual processor thread process IDs are 2457, 2458, 2459, and 2460. Affinitize to target core set 0F000000 (18 core CPU):

```
# taskset -p 1000000 2457
# taskset -p 2000000 2458
# taskset -p 4000000 2459
# taskset -p 8000000 2460
```

In this case, the VM2 virtual processor thread process IDs are 3176, 3177, 3178, and 3179. Affinitize to target core set 0F0000000 (18 core CPU):

```
# taskset -p 10000000 3176
# taskset -p 20000000 3177
# taskset -p 40000000 3178
# taskset -p 80000000 3179
```

The remaining QEMU tasks run on the general purpose core set 3C0000, which these general purpose core should not be isolated on the kernel boot line.

I.2 Configuring the VM

The VM will have four virtual cores and should not have any CPU cores isolated on the kernel boot line. The VM was started with all 4K pages by not using a hugepage memory manager.

I.2.1 Verifying Passthrough

Once the guest starts, run the following command within guest:

```
# lspci -nn
```

Passthrough PCI devices should appear with the same description as the host originally showed. For example, if the following was shown on the host:

```
0c:00.0 Co-processor [0b40]: Intel Corporation Coleto Creek PCIe Endpoint [8086:0435]
```

It should show up on the guest as:

```
00:06.0 Co-processor [0b40]: Intel Corporation Coleto Creek PCIe Endpoint [8086:0435]
```

I.2.2 Installing Intel® Communications Chipset Software in KVM Guest

The instructions in this solutions guide assume that you have super user privileges. The QAT build directory used in this section is /QAT:

```
# su
# mkdir /QAT
# cd /QAT
```

1. Download the Intel® QuickAssist Technology driver:

   ```
   # wget https://01.org/sites/default/files/page/qatmux.l.2.2.0-30.tgz
   ```

2. Extract:

   ```
   # tar -zxof qatmux.l.2.2.0-30.tgz
   ```

3. Launch the script using the following command:

   ```
   # ./installer.sh
   ```

4. Choose option 2 to build and install the acceleration software. Choose option 6 to build the sample LKCF code.

5. Start/stop the accelerator software:

   ```
   # service qat_service start/stop
   ```

6. Set up the environment to install the Linux kernel crypto framework driver:

   ```
   # export ICP_ROOT=/QAT /QAT1.6
   # export KERNEL_SOURCE_ROOT=/usr/src/kernels/`uname -r`
   ```

7. Download the Linux kernel crypto driver:

   ```
   # mkdir -p $ICP_ROOT/quickassist/shims/netkey
   # cd $ICP_ROOT/quickassist/shims/netkey
   # wget https://01.org/sites/default/files/page/qat_patches_netkeyshim.zip
   ```

8. Unzip and unpack the Linux kernel crypto driver:

   ```
   # unzip qat_patches_netkeyshim.zip
   # tar -xzof QATpatches_netkeyshim/icp_qat_netkey.L.0.4.2-10.tar.gz
   ```

9. Build the Linux kernel crypto driver:

   ```
   # cd $ICP_ROOT/quickassist/shims/netkey/icp_netkey
   # make
   ```

10. Install the Linux kernel crypto driver:

    ```
    # cd $ICP_ROOT/quickassist/shims/netkey/icp_netkey
    # insmod ./icp_qat_netkey.ko
    ```

11. Verify that the module has been installed:

    ```
    # lsmod | grep icp
    ```

    The following is the expected output (different id numbers):

    ```
    icp_qat_netkey      21868 0
    icp_qa_al     1551435 2 icp_qat_netkey
    ```

Appendix J Glossary

Abbreviation	Description
ASLR	Address Space Layout Randomization
DPDK	Data Plane Development Kit
IOMMU	Input/Output Memory Management Unit
IRQ	Interrupt Request
MAC	Media Access Control
Mpps	Million packets per second
NFS	Network File System
NFV	Network Function Virtualization
NIC	Network Interface Card
NUMA	Non-Uniform Memory Access
OVS	Open vSwitch
PF	Physical Function
pps	Packets per second
QAT	Quick Assist Technology
SDN	Software Defined Network
SR-IOV	Single Root I/O Virtualization
UIO	Userspace IO
UUID	Universally Unique Identifier
VF	Virtual Function
VT-d	Intel® Virtualization Technology for Directed I/O

Appendix K Packet Throughput

There is a difference between an Ethernet frame, an IP packet, and a UDP datagram. In the seven-layer OSI model of computer networking, packet refers to a data unit at layer 3 (network layer). The correct term for a data unit at layer 2 (data link layer) is a frame, and at layer 4 (transport layer) is a segment or datagram.

Important concepts related to 10GbE performance are frame rate and throughput. The MAC bit rate of 10GbE, defined in the IEEE standard 802.3ae, is 10 billion bits per second. Frame rate is based on the bit rate and frame format definitions. Throughput, defined in IETF RFC 1242, is the highest rate at which the system under test can forward the offered load, without loss.

The frame rate for 10 GbE is determined by a formula that divides the 10 billion bits per second by the preamble + frame length + interframe gap.

The maximum frame rate is calculated using the minimum values of the following parameters as described in the IEEE 802.3ae standard:

- Preamble: 8 bytes * 8 = 64 bits
- Frame length: 64 bytes (minimum) * 8 = 512 bits
- Interframe gap: 12 bytes (minimum) * 8 = 96 bits Therefore, Maximum Frame Rate (64-byte packets)

= MAC Transmit Bit Rate / (Preamble + Frame Length + Inter-frame Gap)

= 10,000,000,000 / (64 + 512 + 96)

= 10,000,000,000 / 672

= 14,880,952.38 frame per second (fps)

Table D–1. Maximum Throughput

IP Packet Size (Bytes)	Theoretical Line Rate (fps) (Full Duplex)	Theoretical Line Rate (fps) (Half Duplex)
64	29,761,904	14,880,952
128	16,891,891	8,445,946
256	9,057,971	4,528,986
512	4,699,248	2,349,624
1024	2,394,636	1,197,318
1280	1,923,076	961,538
1518	1,625,487	812,744

K.1 RFC 2544

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.

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

Although not included in the defined RFC 2544 standard, another crucial measurement in Ethernet networking is packet jitter.

Appendix L References

Document Name	Source
Internet Protocol version 4	http://www.ietf.org/rfc/rfc791.txt
Internet Protocol version 6	http://www.faqs.org/rfc/rfc2460.txt
Intel® 82599 10 Gigabit Ethernet Controller Datasheet	http://www.intel.com/content/www/us/en/ethernet-controllers/82599- 10-gbe-controller-datasheet.html
Intel DDIO	https://www-ssl.intel.com/content/www/us/en/io/direct-data-i-o.html?
Bandwidth Sharing Fairness	http://www.intel.com/content/www/us/en/network-adapters/10-gigabit- network-adapters/10-gbe-ethernet-flexible-port-partitioning-brief.html
Design Considerations for efficient network applications with Intel® multi-core processor-based systems on Linux	http://download.intel.com/design/intarch/papers/324176.pdf
OpenFlow with Intel 82599	http://ftp.sunet.se/pub/Linux/distributions/bifrost/seminars/workshop- 2011-03-31/Openflow_1103031.pdf
Wu, W., DeMar, P. and Crawford, M. (2012). "A Transport- Friendly NIC for Multicore/ Multiprocessor Systems"	IEEE transactions on parallel and distributed systems, vol 23, no 4, April 2012. http://lss.fnal.gov/archive/2010/pub/fermilab-pub-10-327-cd.pdf
Why does Flow Director Cause Placket Reordering?	http://arxiv.org/ftp/arxiv/papers/1106/1106.0443.pdf
IA packet processing	http://www.intel.com/p/en_US/embedded/hwsw/technology/packet- processing
High Performance Packet Processing on Cloud Platforms using Linux* with Intel® Architecture	http://networkbuilders.intel.com/docs/network_builders_RA_packet_processing.pdf
Packet Processing Performance of Virtualized Platforms with Linux* and Intel® Architecture	http://networkbuilders.intel.com/docs/network_builders_RA_NFV.pdf
DPDK	http://www.intel.com/go/dpdk
RFC 1242 (Benchmarking Terminology for Network Interconnection Devices)	http://www.ietf.org/rfc/rfc1242.txt
RFC 2544 (Benchmarking Methodology for Network Interconnect Devices)	http://www.ietf.org/rfc/rfc2544.txt
RFC 6349 (Framework for TCP Throughput Testing)	http://www.faqs.org/rfc/rfc6349.txt
RFC 3393 (IP Packet Delay Variation Metric for IP Performance Metrics)	https://www.ietf.org/rfc/rfc3393.txt
MEF end-end measurement metrics	http://metroethernetforum.org/Assets/White_Papers/Metro-Ethernet- Services.pdf
DPDK Sample Application User Guide	http://dpdk.org/doc/intel/dpdk-sample-apps-1.7.0.pdf
Benchmarking Methodology for Virtualization Network Performance	https://tools.ietf.org/html/draft-liu-bmwg-virtual-network-benchmark-00
Benchmarking Virtual Network Functions and Their Infrastructure	https://tools.ietf.org/html/draft-morton-bmwg-virtual-net-01

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.