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.

A “Mission Critical Application” is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL’S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENCE IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

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.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm.

Intel and Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2014, Intel Corporation. All Rights Reserved.
## Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>September 18, 2014</td>
<td>Minor edits throughout document.</td>
</tr>
</tbody>
</table>
Contents

1.0 Audience and Purpose ................................................................. 7
2.0 Summary ..................................................................................... 9
2.1 Recommended Setup and Validation Procedures ...................... 11
2.2 Network Services Examples ..................................................... 12
2.2.1 Suricata (Next Generation IDS/IPS engine) ......................... 12
2.2.2 vBNG (Broadband Network Gateway) ............................... 12
3.0 Hardware Components .............................................................. 15
4.0 Software Versions ...................................................................... 17
4.1 Obtaining Software Ingredients ................................................. 18
5.0 Installation and Configuration Guide .......................................... 19
5.1 Instructions Common to Compute and Controller Nodes .......... 19
5.1.1 BIOS Settings .................................................................. 19
5.1.2 Operating System Installation and Configuration ................. 20
5.2 Controller Node Setup ............................................................. 22
5.2.1 OpenStack (Icehouse) ..................................................... 22
5.2.2 OpenDaylight (Hydrogen) ............................................... 26
5.3 Compute Node Setup ............................................................... 29
5.3.1 Host Configuration ......................................................... 29
6.0 Testing the Controller Node ....................................................... 35
6.1 Preparation with OpenStack .................................................... 35
6.1.1 Deploying Virtual Machines ............................................ 35
6.2 Using OpenDaylight ............................................................... 43
6.2.1 Preparing OpenDaylight to Connect with the Compute Host .... 43
6.2.2 Connecting OpenDaylight to the Compute Host .................. 43
6.2.3 Configuring OpenFlow Rules Using OpenDaylight ............. 46
7.0 Testing Compute Node Performance .......................................... 53
7.1 Intel® DPDK and Linux Kernel L3 Forwarding on the Host .......... 53
7.1.1 L3 Forwarding Performance Characterization .................... 54
7.2 Port-Port Forwarding on Host Using Intel® DPDK Accelerated Open vSwitch and Open vSwitch ....... 55
7.2.1 Port-port vSwitch Performance Characterization ................ 56
7.3 L3 Forwarding in the VM Using Open vSwitch (OVS) and DPDK vSwitch ........................................ 56
7.3.1 Single VM L3 Forwarding Performance Characterization .... 57
7.4 L3 Forwarding in Two VMs Using Open vSwitch (OVS) and DPDK vSwitch with userspace vHost ......... 58
7.4.1 Two VMs L3 Forwarding Performance Characterization ...... 59
7.4.2 Scalability Tests for Two VMs Using userspace vHost as vNICs 59
7.5 VM-VM IPsec Tunnel Performance Using Quick Assist Technology ... 60
Appendix A  Test Methodology............................................................ 63
A.1 Defining Packet Throughput ...................................................... 63
A.2 Network Protocols ................................................................. 63
A.3 RFC2544 ............................................................................. 63
A.4 Netperf ................................................................................. 63
Appendix B  Installing and Running DPDK vSwitch on the CLI................. 65
B.1 Intel DPDK 1.7 with Fedora 20 ............................................... 65
B.1.1 Building a DPDK Target from Source .............................. 65
B.1.2 Hugepage Configuration ................................................ 65
B.1.3 Loading DPDK igb_uio and Binding the Drivers to 10 GbE Interfaces ............................................ 66
B.2 Guest Operating System Installation and Configuration ............. 67
1.0 Audience and Purpose

The primary audiences for this document are architects and engineers implementing the Intel® Open Network Platform Server Reference Architecture using Open Source software. Software ingredients include:

- DevStack*
- OpenStack*
- OpenDaylight*
- Intel® Data Plane Development Kit (Intel® DPDK)
- Intel® DPDK vSwitch
- Open vSwitch*
- Fedora 20*

This document provides a guide for integration and performance characterization using the Intel® Open Network Platform Server. Content includes high-level architecture, setup and configuration procedures, integration learnings, and a set of baseline performance data. This information is intended to help architects and engineers evaluate Network Function Virtualization (NFV) and Software Defined Network (SDN) solutions.

An understanding of system performance is required to develop solutions that meet the demanding requirements of the telecom industry and transform telecom networks. Workload examples are described and are useful for evaluating other NFV workloads.

Ingredient versions, integration procedures, configuration parameters, and test methodologies all influence performance. The performance data provided here does not represent best possible performance, but rather provides a baseline of what is possible using “out-of-box” open source software ingredients.

The purpose of documenting configurations is not to imply any preferred methods. However, providing a baseline configuration of well tested procedures can help to achieve optimal system performance when developing an NFV/SDN solution.
NOTE: This page intentionally left blank.
2.0 Summary

The Intel® Open Network Platform Server uses Open Source software to help accelerate SDN and NFV commercialization with the latest Intel Architecture Communications Platform.

This document describes how to setup and configure controller and compute nodes for evaluating and developing NFV/SDN solutions using the Intel® Open Network Platform ingredients.

Platform hardware is based on a Xeon® DP Server with the following:

- Intel® Xeon® Processor Series E5-2680 v2
- Intel® 82599 10 GbE Controller
- Intel® Communications Chipset 8920 with Intel Quick Assist Technology.

The host operating system is Fedora® 20 with Qemu-kvm virtualization technology. Software ingredients include Intel® Data Plane Development Kit (Intel® DPDK), Open vSwitch, Intel® DPDK vSwitch, OpenStack, and OpenDaylight.

![Intel® Open Network Platform Server - Hardware and Software Ingredients](image-url)
In addition, Appendix B describes how to use DevStack (http://devstack.org/) to deploy compute and controller nodes, and provides information on how to manually set up compute nodes. The two deployment processes are procedurally different and should not be confused. Describing how to build the compute node manually has the benefit of illustrating performance optimization methods, and familiarizing the reader with what happens “under the hood.” All the performance results in this document were captured using the manual deployment procedure.

Figure 2-2 shows a generic SDN/NFV setup. In this configuration, Orchestrator and Controller (management and control plane) and compute node (data plane) run on different server nodes. Note that many variations of this setup can be deployed.

The test cases described in this document were designed to illustrate certain baseline performance and functionality using the specified ingredients, configurations, and specific test methodology. A simple network topology was used, as shown in Figure 2-2.

Test cases are designed to:

- Baseline packet processing (such as data plane) performance with host and VM configurations.
- Verify communication between controller and compute nodes.
- Validate basic controller functionality.
- Compare Open vSwitch OVS and Intel® DPDK Accelerated vSwitch performance.
2.1  Recommended Setup and Validation Procedures

**Compute node:**

1. BIOS settings.
2. Host OS (Fedora 20).
   - Port-to-port L3 forwarding performance using Linux* kernel.
3. DPDK configuration.
   - Port-to-port L3 forwarding performance using Intel® DPDK.
4. vSwitch and OpenStack components.
   - Switching performance using Intel® DPDK vSwitch (port-to-port network throughput).
   - Switching performance using OVS (port-to-port network throughput).
5. Virtual Machine
   - Port-to-port L3 forwarding performance in VM using OVS (1 VM and 2 VM results provided).
   - Port-to-port L3 forwarding performance in VM using Intel® DPDK (1 VM and 2 VM results provided).
6. Quick Assist.
   - QAT performance tests.
7. Suricata IPS.
   - Functional tests.

**Orchestrator/Controller node:**

1. BIOS settings.
2. Host OS (Fedora 20).
3. OpenStack (Icehouse) – Orchestrator.
4. OpenDaylight (Hydrogen) – Network Controller.
   - Bringing up and configuring a VM in the Compute Node.

Certain relative performance results are summarized in Table 2-1 through Table 2-3. Unless otherwise indicated, performance data was obtained using RFC2544 “Benchmarking Methodology for Network Interconnect Devices” (https://www.ietf.org/rfc/rfc2544.txt). For details re. packet throughput, protocols used and test methodology refer to Appendix A (Test Methodology).

### Table 2-1  Host Operating System Performance Summary

<table>
<thead>
<tr>
<th>Host Operating System Test Case</th>
<th>Linux kernel</th>
<th>Intel® DPDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port-to-port L3 forwarding performance</td>
<td>x</td>
<td>14x</td>
</tr>
</tbody>
</table>

1. Relative performance is based on 64-byte Ethernet frames using RFC2544 zero loss methodology. Refer to Appendix A for more details.
2.2 Network Services Examples

The following examples of network services are included as use-cases that have been tested with the Intel® Open Network Platform Server Reference Architecture.

2.2.1 Suricata (Next Generation IDS/IPS engine)

Suricata is a high performance Network IDS, IPS, and Network Security Monitoring engine developed by the OISF, its supporting vendors, and the community.

http://suricata-ids.org/

2.2.2 vBNG (Broadband Network Gateway)

Intel Data Plane Performance Demonstrators – Border Network Gateway (BNG) using Intel® DPDK.


A Broadband (or Border) Network Gateway may also be known as a Broadband Remote Access Server (BRAS) and routes traffic to and from broadband remote access devices, such as digital subscriber line access multiplexers (DSLAM). This network function is included as an example of a workload that can be virtualized on the Intel® Open Network Platform Server.
Additional information on the performance characterization of this vBNG implementation can be found at:

http://networkbuilders.intel.com/docs/Network_Builders_RA_vBRAS_Final.pdf

Refer to Appendix C for information on setting up and testing the vBNG application.
## 3.0 Hardware Components

### Table 3-1 Hardware Ingredients

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Intel® Server Board 2U 8x3.5 SATA 2x750W 2xHS Rails Intel R2308GZ4GC</td>
<td>Grizzly Pass Xeon DP Server (2 CPU sockets). 240GB SSD 2.5in SATA 6Gb/s Intel Wolfsville SSDSC2BB240G401 DC S3500 Series</td>
</tr>
<tr>
<td>Processors</td>
<td>Intel® Xeon® Processor Series E5-2680 v2 LGA2011 2.8GHz 25MB 115W 10 cores</td>
<td>Ivy Bridge Socket-R (EP), 10 Core, 2.8GHz, 115W, 2.5M per core LLC, 8.0 GT/s QPI, DDR3-1867, HT, turbo Long product availability</td>
</tr>
<tr>
<td>Cores</td>
<td>10 physical cores/CPU</td>
<td>20 Hyper-threaded cores per CPU for 40 total cores</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB 1600 Reg ECC 1.5 V DDR3 Kingston KVR16R1154/8I Romley</td>
<td>64 GB RAM (8x 8 GB)</td>
</tr>
<tr>
<td>NICs</td>
<td>2x Intel® 82599 10 GbE Controller (Niantic)</td>
<td>NICs are on socket zero (3 PCIe slots available on socket 0)</td>
</tr>
<tr>
<td>BIOS</td>
<td>SE5C600.86B.02.01.0002.082220131453 Release Date: 08/22/2013 BIOS Revision: 4.6</td>
<td>Intel® Virtualization Technology for Directed I/O (Intel® VT-d) enabled only for QAT tests Hyper-Threading disabled</td>
</tr>
<tr>
<td>Quick Assist Technology</td>
<td>Intel® Communications Chipset 8920 (Coleto Creek)</td>
<td>Walnut Hill PCIe card 1x Coleto Creek</td>
</tr>
</tbody>
</table>
# 4.0 Software Versions

## Table 4-1 Software Versions

<table>
<thead>
<tr>
<th>Software Component</th>
<th>Function</th>
<th>Version/Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fedora 20 x86_64</td>
<td>Host OS</td>
<td>3.15.6-200 kernel</td>
</tr>
<tr>
<td>Qemu-kvm</td>
<td>Virtualization technology</td>
<td>Modified QEMU 1.6.2 (bundled with Intel® DPDK vSwitch)</td>
</tr>
<tr>
<td>Intel® Data Plane Development Kit (Intel® DPDK)</td>
<td>Network Stack bypass</td>
<td>Intel® DPDK 1.7.0</td>
</tr>
<tr>
<td>Intel® DPDK vSwitch</td>
<td>vSwitch</td>
<td>v1.1.0</td>
</tr>
<tr>
<td>Open vSwitch</td>
<td>SDN Orchestrator</td>
<td>Icehouse Release + Intel patches (openstack_0vdk.i.0.2-902.zip)</td>
</tr>
<tr>
<td>DevStack</td>
<td>Tool for Open Stack deployment</td>
<td>git clone <a href="https://github.com/openstack-dev/devstack.git">https://github.com/openstack-dev/devstack.git</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>git checkout ea548cd5ce1c0899bc12cd2ff957546ecb7e88b9</td>
</tr>
<tr>
<td>OpenDaylight</td>
<td>SDN Controller</td>
<td>Hydrogen Release</td>
</tr>
<tr>
<td>Quick Assist and NetKey shim</td>
<td>Crypto Accelerator</td>
<td>QATmux.L.1.1.0-60.tar.gz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>icp_qat_netkey.L.0.4.2-10.tar.gz</td>
</tr>
<tr>
<td>Suricata</td>
<td>IPS application</td>
<td>Suricata v2.0.2 (current Fedora 20 package)</td>
</tr>
<tr>
<td>strongSwan</td>
<td>IPSec stack</td>
<td>strongSwan v.4.5.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://download.strongswan.org/strongswan-4.5.3.tar.gz">http://download.strongswan.org/strongswan-4.5.3.tar.gz</a></td>
</tr>
<tr>
<td>BNG DPPD</td>
<td>Broadband Network Gateway DPDK Performance Demonstrator Application</td>
<td>DPPD v011</td>
</tr>
<tr>
<td>PktGen</td>
<td>Software Network Package Generator</td>
<td>v.2.7.1</td>
</tr>
</tbody>
</table>
### 4.1 Obtaining Software Ingredients

#### Table 4-2 Software Ingredients

<table>
<thead>
<tr>
<th>Software Component</th>
<th>Software Sub-components</th>
<th>Patches</th>
<th>Location</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fedora 20</td>
<td></td>
<td></td>
<td><a href="http://download.fedoraproject.org/pub/fedora/linux/releases/20/Fedora/x86_64/iso/Fedora-20-x86_64-DVD.iso">http://download.fedoraproject.org/pub/fedora/linux/releases/20/Fedora/x86_64/iso/Fedora-20-x86_64-DVD.iso</a></td>
<td>Standard Fedora 20 iso image.</td>
</tr>
<tr>
<td>Intel® Data Plane Development Kit (Intel® DPDK)</td>
<td>DPDK poll mode driver, sample apps (bundled)</td>
<td><a href="http://dpdk.org/download">http://dpdk.org/download</a></td>
<td>All sub-components in one zip file.</td>
<td></td>
</tr>
<tr>
<td>DPDK vSwitch</td>
<td>dpdk-ovs, qemu, ovs-db, vswitchd, ovs_client (bundled)</td>
<td><a href="https://github.com/01org/dpdk-ovs/releases">https://github.com/01org/dpdk-ovs/releases</a></td>
<td>v1.1.0</td>
<td></td>
</tr>
<tr>
<td>Open vSwitch</td>
<td></td>
<td>Package from Fedora 20 using yum install Open vSwitch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenStack</td>
<td>Patches for Nova, Neutron and DevStack</td>
<td>Icehouse release. To be deployed using: <a href="https://01.org/sites/default/files/page/openstack_ovdk.l.0.2-902.zip">https://01.org/sites/default/files/page/openstack_ovdk.l.0.2-902.zip</a></td>
<td>Three patches downloaded as one tarball. Then follow the instructions to deploy the Nodes.</td>
<td></td>
</tr>
<tr>
<td>Intel® ONP Server Release 1.1 Script</td>
<td>Helper scripts to setup SRT 1.1 using devstack</td>
<td><a href="https://01.org/sites/default/files/page/onps_11.tgz">https://01.org/sites/default/files/page/onps_11.tgz</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QAT</td>
<td>QAT Driver</td>
<td></td>
<td><a href="https://01.org/sites/default/files/page/qatmux.l.1.1.0-60.tar.gz">https://01.org/sites/default/files/page/qatmux.l.1.1.0-60.tar.gz</a></td>
<td>Two patches need to be applied against QAT1.6</td>
</tr>
<tr>
<td></td>
<td>NetKey shim (not bundled)</td>
<td>Intel Quickassist Technology patch for Netkey driver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PktGen</td>
<td>Software Network Package Generator</td>
<td><a href="https://github.com/Pktgen/Pktgen-DPDK">https://github.com/Pktgen/Pktgen-DPDK</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNG Helper scripts</td>
<td>Intel® ONP for Server Configuration Scripts for vBNG</td>
<td><a href="https://01.org/sites/default/files/page/vbng-scripts.zip">https://01.org/sites/default/files/page/vbng-scripts.zip</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suricata</td>
<td></td>
<td>Package from Fedora 20.</td>
<td>yum install suricata</td>
<td></td>
</tr>
<tr>
<td>strongSwan*</td>
<td></td>
<td></td>
<td><a href="http://download.strongswan.org/strongswan-4.5.3.tar.gz">http://download.strongswan.org/strongswan-4.5.3.tar.gz</a></td>
<td></td>
</tr>
</tbody>
</table>
5.0 Installation and Configuration Guide

This section describes the installation and configuration instructions to prepare the controller and compute nodes.

5.1 Instructions Common to Compute and Controller Nodes

This section describes how to prepare both the controller and compute nodes with the right BIOS settings and operating system installation. The preferred operating system is Fedora 20, although it is considered relatively easy to use this solutions guide for other Linux distributions.

5.1.1 BIOS Settings

Table 5-1 BIOS Settings

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Setting for Controller Node</th>
<th>Setting for Compute Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhanced Intel SpeedStep</td>
<td>Enabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>Processor C3</td>
<td>Disabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>Processor C6</td>
<td>Disabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>Intel Virtualization Technology for Directed I/O (Vt-d)</td>
<td>Disabled</td>
<td>Disabled (Enabled for QAT)</td>
</tr>
<tr>
<td>Intel Hyper-Threading Technology (HTT)</td>
<td>Enabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>MLC Streamer</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>MLC Spatial Prefetcher</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DCU Instruction Prefetcher</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Direct Cache Access (DCA)</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>CPU Power and Performance Policy</td>
<td>Performance</td>
<td>Performance</td>
</tr>
<tr>
<td>Intel Turbo boost</td>
<td>Enabled</td>
<td>Off</td>
</tr>
<tr>
<td>Memory RAS and Performance Configuration -&gt; Numa Optimized</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
5.1.2 Operating System Installation and Configuration

Following are some generic instructions for installing and configuring the operating system. Other ways of installing the operating system are not described in this solutions guide, such as network installation, PXE boot installation, USB key installation, etc.

5.1.2.1 Getting the Fedora 20 DVD

1. Download the 64-bit Fedora 20 DVD (not Fedora 20 Live Media) from the following site:
   http://fedoraproject.org/en/get-fedora#formats
   or from direct URL:
   http://download.fedoraproject.org/pub/fedora/linux/releases/20/Fedora/x86_64/iso/Fedora-20-x86_64-DVD.iso

2. Burn the ISO file to DVD and create an installation disk.

5.1.2.2 Fedora 20 Installation

Use the DVD to install Fedora 20. During the installation, click Software selection, then choose the following:

1. C Development Tool and Libraries
2. Development Tools

Also create a user stack and check the box Make this user administrator during the installation. The user stack is used in OpenStack installation.

5.1.2.3 Additional Packages Installation

Some of the packages are not supplied with Fedora 20, but are required by Intel® Open Network Platform Software components. They should be installed by the user.

The discussions that follow list the packages required in the controller and compute nodes. Note that there is a distinction in the requirements between the host and guest in the compute node.

Furthermore, the following discussions list the Fedora packages that also need to be installed. Some of them might have been installed during Fedora 20 installation, depending on the method that was used to perform the installation.

5.1.2.3.1 Fedora Packages Needed for the Controller Node

    ntp patch socat python-passlib libxslt-devel libffi-devel fuse-devel kernel-modules-extra git zip screen gcc automake

5.1.2.3.2 Fedora Packages Needed for the Compute Node (Host)

5.1.2.3.3  Fedora Packages Needed for the Compute Node (Guest)

rpmdevtools yum-utils ncurses-devel qt3-devel libXi-devel gcc-c++ openssl-devel glibc.i686 libgcc.i686 libstdc++.i686 libcap-devel gcc kernel-devel coreutils make nasm glibc-devel glibc-devel.i686.6 autoconf zlib-devel glib2-devel libtool fuse-devel boost-devel gmp-devel automake icp_qat_netkey.L0.4.2, pixman-devel, strongswan4.5.3, & QAT 1.6

For compute nodes, huge pages are required (see Appendix B.1.2 for details).

5.1.2.3.4  Fedora 20 Operating System Update

Note: The Open Networking Platform Software depends on libraries provided by your Linux distribution. As such, it is recommended that you regularly update your Linux distribution with the latest bug fixes and security patches to reduce the risk of security vulnerabilities in your systems.

After installing the required packages, the operating system should be updated with the following command:

    yum update -y

This command will upgrade to the latest kernel that Fedora supports. Afterwards, reboot the system.

5.1.2.4  Disable and Enable Services

For OpenStack, the following services were disabled: selinux, firewall, and NetworkManager. Run the following commands:

    sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
    systemctl disable firewalld.service
    systemctl disable NetworkManager.service
    systemctl disable irqbalance.service

Another more secure option might be to disable the firewall to just configuring it to enable access to and/or from the services that are accessing the system.

The following services should be enabled: ntp, sshd, and network. Run the following commands:

    systemctl enable ntpd.service
    systemctl enable ntpdate.service
    systemctl enable sshd.service
    chkconfig network on

It is important to keep the timing synchronized between all nodes. It is also necessary to use a known NTP server for all nodes. The user can edit etc/ntp.conf to add a new server and remove default servers. The following example replaces a default NTP server with a local NTP server 10.0.0.12 and comments out other default servers

    sed -i 's/server 0.fedora.pool.ntp.org iburst/server 10.0.0.12/g' /etc/ntp.conf
    sed -i 's/server 1.fedora.pool.ntp.org iburst/# server 1.fedora.pool.ntp.org iburst /g' /etc/ntp.conf
    sed -i 's/server 2.fedora.pool.ntp.org iburst/# server 2.fedora.pool.ntp.org iburst /g' /etc/ntp.conf
    sed -i 's/server 3.fedora.pool.ntp.org iburst/# server 3.fedora.pool.ntp.org iburst /g' /etc/ntp.conf
5.2 Controller Node Setup

This section describes the controller node setup. It is assumed that the user successfully followed the BIOS settings and the operating system installation and configuration sections.

5.2.1 OpenStack (Icehouse)

This section documents features and limitations that are supported with Intel® DPDK vSwitch and OpenStack Icehouse.

SRT 1.1 uses patches for the operating system.

5.2.1.1 Network Requirements

General

At least two networks are required to build OpenStack infrastructure in a lab environment. One network is used to connect all nodes for OpenStack management, and the other one is a private network, exclusively used by OpenStack internal connection between all instances (or virtual machines).

One additional network is required for Internet connectivity, as installing OpenStack requires pulling packages from various sources/repositories on the Internet.

Some users might want to have Internet and/or external connectivity for OpenStack instances (virtual machines). In this case, an optional network can be used.

The assumption is that the targeting OpenStack infrastructure contains multiple nodes; one is controller node and one or more are compute node(s).

Network Configuration Example

The following is an example of how to configure networks for OpenStack infrastructure. The example uses four network interfaces as follows:

- ens2f1: Internet network - Used to pull all necessary packages/patches from repositories on the Internet and obtain a DHCP address.
- ens2f0: Management network - Used to connect all nodes for OpenStack management (use network 10.11.0.0/16).
- p1p1: Virtual network - Used for OpenStack internal connections for virtual machines (no IP address).
- p1p2: Optional external network - Used for virtual machine Internet/external connectivity (no IP address). This interface is only in the Controller node if external network is configured. For Compute node, this interface is not needed.

Note that, among these interfaces, interface for virtual network (in this example, p1p1) must be an 82599 port because it is used for Intel® DPDK and DPDK vSwitch. Also note that a static IP address should be used for interface of management network.

In Fedora 20, the network configuration files are located at:

```
/etc/sysconfig/network-scripts/
```

To configure a network on the host system, edit the following network configuration files:
ifcfg-ens2f1
    DEVICE=ens2f1
    TYPE=Ethernet ONBOOT=yes BOOTPROTO=dhcp

ifcfg-ens2f0
    DEVICE=ens2f0
    TYPE=Ethernet
    ONBOOT=yes
    BOOTPROTO=static
    IPADDR=10.11.12.11
    NETMASK=255.255.0.0

ifcfg-p1p1
    DEVICE=p1p1
    TYPE=Ethernet ONBOOT=yes BOOTPROTO=none

ifcfg-p1p2
    DEVICE=p1p2
    TYPE=Ethernet ONBOOT=yes BOOTPROTO=none

Note: Do not configure the IP address for p1p1 (10 Gb/s interface); otherwise, Intel® DPDK will not work when binding the driver during OpenStack Neutron installation.

Note: 10.11.12.11 and 255.255.0.0 are static IP address and net mask to the management network. It is necessary to have static IP address on this subnet. The IP address 10.11.12.11 is just an example.

5.2.1.2 Storage Requirements

By default, DevStack uses blocked storage (Cinder) with a volume group, stack-volumes. If not specified, stack-volumes is created with 10 Gb/s space from a local file system. Note that stack-volumes is the name for the volume group, not more than 1 volume.

The following example shows how to use spare local disks, /dev/sdb and /dev/sdc, to form stack-volumes on a controller node by running the following commands:

    pvcreate /dev/sdb
    pvcreate /dev/sdc
    vgcreate stack-volumes /dev/sdb /dev/sdc

5.2.1.3 OpenStack Installation Procedures

General

DevStack is used to deploy OpenStack in this example. The following procedure uses an actual example of an installation performed in an Intel test lab, consisting of one controller node (controller) and one compute node (compute).

Controller Node Installation Procedures

The following example uses a host for controller node installation with the following:

- Hostname: sdnlab-k01
- Internet network IP address: Obtained from DHCP server
- OpenStack Management IP address: 10.11.12.1
- User/password: stack/stack
Root User Actions

Login as **su** or **root** user and perform the following:

1. Add stack user to `sudoer` list

   ```
   echo "stack ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
   ```

2. Edit `/etc/libvirt/qemu.conf`, add or modify with the following lines:

   ```
   cgroup_controllers = [ "cpu", "devices", "memory", "blkio", "cpuset","cpuacct" ]
   cgroup_device_acl = [ 
       "/dev/null", "/dev/full", "/dev/zero", 
       "/dev/random", "/dev/urandom", 
       "/dev/ptmx", "/dev/kvm", "/dev/kqemu", 
       "/dev/rtc", "/dev/hpet", "/dev/net/tun", 
       "/mnt/huge", "/dev/vhost-net"
   ]
   hugetlms_mount = "/mnt/huge"
   ```

3. Restart `libvirt` service and make sure `libvird` is active

   ```
   systemctl restart libvirtd.service
   systemctl status libvirtd.service
   ```

Stack User Actions

1. Login as a stack user.

2. Configure the appropriate proxies (yum, http, https, and git) for package installation, and make sure these proxies are functional. Note that on controller node, localhost and it's IP address should be included in no_proxy setup (for example, ```export no_proxy=localhost,10.11.12.1```).

3. Intel® DPDK vSwitch patches for OpenStack.

   The tar file `openstack-ovdk.l.0.2-902.zip` contains necessary patches for OpenStack. Currently it is not native to the OpenStack. The file can be downloaded from:

   ```
   https://01.org/sites/default/files/page/openstack_ovdk.l.0.2-902.zip
   ```

   Place the file in the `/home/stack/` directory and unzip. Three patch files: `devstack.patch`, `nova.patch`, and `neutron.patch`, will be present after unzip.

   ```
   cd /home/stack
   wget https://01.org/sites/default/files/page/openstack_ovdk.l.0.2-902.zip
   unzip openstack.ovdk.l.0.2-902.zip
   ```

4. Download DevStack source.

   ```
   git clone https://github.com/openstack-dev/devstack.git
   ```

5. Check out DevStack with Intel® DPDK vSwitch and patch.

   ```
   cd /home/stack/devstack/
   git checkout ea548cd5ce1c0899bc12cd2ff957546ecb7e88b9
   patch -p1 < /home/stack/devstack.patch
   ```

6. Download and patch Nova and Neutron.

   ```
   sudo mkdir /opt/stack
   sudo chown stack:stack /opt/stack
   cd /opt/stack/
   git clone https://github.com/openstack/nova.git
   git clone https://github.com/openstack/neutron.git
   cd /opt/stack/nova/
   git checkout 2014.1.1
   patch -p1 < /home/stack/nova.patch
   cd /opt/stack/neutron/
   git checkout 2014.1.1
   ```
patch -p1 < /home/stack/neutron.patch

7. Create `local.conf` file in `/home/stack/devstack/`

8. Pay attention to the following in the `local.conf` file:

   a. Use Rabbit for messaging services (Rabbit is on by default). In the past, Fedora only supported QPID for OpenStack. Now it only supports Rabbit.

   b. Do not install compute on the controller, and disable Nova compute service.

```
   disable n-cpu
```

   c. To use Open vSwitch, specify in configuration for ML2 plug-in.

```
   Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch
```

   d. A sample `local.conf` files for controller node is as follows:

```
# Controller node
[[local|localrc]]

FORCE=yes
ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
DATABASE_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=no-token-password
HORIZON_PASSWORD=password
RABBIT_PASSWORD=password

disable_service n-net
disable_service n-cpu
enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
enable_service neutron
enable_service horizon

Q_AGENT=openvswitch
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch
Q_ML2_PLUGIN_TYPE_DRIVERS=vlan,flat,local

DEST=/opt/stack
LOGFILE=$DEST/stack.sh.log
SCREEN_LOGDIR=$DEST/screen
SYSLOG=True
LOGDAYS=1
HOST_IP_IFACE=ens2f0
PUBLIC_INTERFACE=p1p2
VLAN_INTERFACE=p1p1
FLAT_INTERFACE=p1p1

ENABLE_TENANT_VLANS=True
PHYSICAL_NETWORK=physnet1
ML2_VLAN_RANGES=physnet1:1000:1010
OVS_PHYSICAL_BRIDGE=br-p1p1
MULTI_HOST=True

[[post-config|$NOVA_CONF]]

[DEFAULT]

firewall_driver=nova.virt.firewall.NoopFirewallDriver
novncproxy_host=0.0.0.0
novncproxy_port=6080
9. Install DevStack

```bash
cd /home/stack/devstack/
./stack.sh
```

10. For a successful installation, the following shows at the end of screen output:

```
stack.sh completed in XXX seconds
```

where XXX is the number of seconds.

11. For controller node only — Add physical port(s) to the bridge(s) created by the DevStack installation. The following example can be used to configure the two bridges: br-p1p1 (for virtual network) and br-ex (for external network).

```
sudo ovs-vsctl add-port br-p1p1 p1p1
sudo ovs-vsctl add-port br-ex p1p2
```

12. Make sure proper VLANs are created in the switch connecting physical port p1p1. For example, the previous `local.conf` specifies VLAN range of 1000-1010; therefore matching VLANs 1000 to 1010 should be configured in the switch.

### 5.2.2 OpenDaylight (Hydrogen)

The OpenDaylight controller is an SDN controller platform that can be used to manage a variety of network devices. For this solutions guide, OpenDaylight is used to manage an OVS host via the OVSDB and OpenFlow protocols. The primary goal is to show how OpenFlow can be used to manually create and manage OpenFlow rules. The use of OpenDaylight to instantiate vSwitch bridges and ports (while possible) is not addressed. The assumption is that vSwitch bridges and ports are configured directly on the compute host using OpenStack.

```
/op
```

**Note:** The OpenDaylight Hydrogen release can be obtained from OpenDaylight Downloads. Instructions for installing two different ways (downloaded zip file or RPM file) are also included at the OpenDaylight downloads website.

Following are instructions for installing and running via RPM. These instructions are applicable for Fedora 20:

1. Download the repository file.

```
```

2. Install the repository file.

```
# sudo rpm -Uvh opendaylight-release-0.1.0-2.fc19.noarch.rpm
```


```
# sudo yum install opendaylight-virtualization
```


   a. Edit the OpenDaylight system configuration file.

```
# sudo vi /etc/sysconfig/opendaylight-controller
```

   b. Configure the correct ODL edition (ODL_DIST).

```
ODL_DIST="virt-ovsdb"
```

   c. Configure extra runtime options (ODL_OPTS).

```
ODL_OPTS="-XX:MaxPermSize=384m"
```
5. Fix OpenFlow version.
   a. Edit the OpenDaylight configuration file.
      
      # vi /etc/opendaylight-controller/config.ini
   b. Ensure ovsdb.of.version setting is configured as follows:
      ovsdb.of.version=1.0

6. Disable the OVSDB Neutron plug-in.

The OVSDB neutron plug-in is basically an OpenDaylight application built into the controller framework that attempts to set things up for working with OpenStack. Thus, it tries to create extra bridge instances used by OpenStack for virtual networking such as br-int, br-tun and br-ext. For SRT 1.1, Intel does not want these bridge instances created, so the solution is to remove the OVSDB Neutron plug-in.

   cd /usr/share/opendaylight-controller/plugins
   sudo rm org.opendaylight.ovsdb.ovsdb.neutron-0.5.0.jar

**Note:** Instead of removing the plug-in file, it is advisable to simply copy to another directory (renaming the file in place by adding a .bak to the filename did not prevent the plug-in from starting.)

5.2.2.1 Starting the OpenDaylight Controller

The simplest method of starting the controller is to use the systemctl system and service manager (see the systemctl man page for more details).

To get the controller running, use systemctl to enable the service and then start it.

- Enable the OpenDaylight service:
  - # sudo systemctl enable opendaylight-controller.service
- Start the OpenDaylight controller (after it has been enabled):
  - # sudo systemctl start opendaylight-controller.service

Following are some additional systemctl commands for managing the OpenDaylight controller.

- Disable the OpenDaylight service:
  - # sudo systemctl disable opendaylight-controller.service
- Stop the OpenDaylight controller:
  - # sudo systemctl stop opendaylight-controller.service
- Check the status of the OpenDaylight service:
  - # sudo systemctl status opendaylight-controller.service

5.2.2.1.1 Interacting with the OpenDaylight Controller

There are several methods of interacting with the OpenDaylight controller. These methods include the OpenDaylight Web GUI, the OpenDaylight OSGi console and via applications, which use OpenDaylight’s REST API.
5.2.2.1.2 The OpenDaylight Web GUI

The OpenDaylight controller can be accessed via a web-based GUI. To access, point a web browser to:

http://<IP address or hostname>:8080

The following login screen displays, as shown in Figure 5-1.

The default username and password is admin.

Once logged in, the main screen displays, as shown in Figure 5-2.

Note: In this screen, the controller has no network elements (such as switches) to manage.
5.2.2.1.3 The OSGi Console

The OpenDaylight controller is implemented as a number of bundles (like Java .jar files) running in an OSGi framework. Sometimes it can be useful to have access to the OSGi console. When the controller starts as a system service, the OSGi console can be accessed via telnet:

```
telnet 127.0.0.1 2400
```

To leave the OSGi console, enter **Ctrl+]** and then type **quit** at the telnet prompt.

5.2.2.1.4 The OpenDaylight REST API

A critical feature of SDN controllers is a northbound API that can be used by other applications to manage the network devices controlled by the SDN controller. OpenDaylight provides a REST API that can be used to access and manage OVS hosts and vSwitches. For the purposes of testing, the REST API and general looking around, an HTTP tool such as Postman for the Chrome browser are very handy. Postman enables a user to construct HTTP requests and send them to the designated server. Useful commands can be saved into collections so that subsequent usage of specific API calls is quick and easy.

5.3 Compute Node Setup

This section describes how to complete the setup of the compute nodes. It is assumed that the user has successfully completed the BIOS settings and operating system installation and configuration sections.

5.3.1 Host Configuration

5.3.1.1 Isolate Cores

On the host and the VM, isolate all the cores except core 0. Core 0 is used to run the operating system and related tasks. Since Grizzly Pass has 20 physical cores (10 in each CPU socket), edit the `/boot/grub2/grub.cfg` file to include the following parameter, and restart the system for it to take effect.

```
isolcpus=1,2 ... 39
```

By running the top command it can be verified that except for CPU0 all CPUs are 0% loaded.

5.3.1.2 Using DevStack to Deploy vSwitch and OpenStack Components

General

Deploying OpenStack and Intel® DPDK vSwitch using DevStack on a compute node follows the same procedures as on the controller node. Differences include:

- Required services are nova compute, neutron agent, and Rabbit.
- Intel® DPDK vSwitch is used in place of Open vSwitch for neutron agent.
- A huge pages configuration is required (see Appendix B.1.2). Note that for OpenStack, huge configuration goes through the `local.conf` file.
Compute Node Installation Example

The following example uses a host for compute node installation with the following:

- Hostname: sdnlab-k02
- Lab network IP address: Obtained from DHCP server
- OpenStack Management IP address: 10.11.12.2
- User/password: stack/stack

Note the following:

- No_proxy setup: Localhost and its IP address should be included in the no_proxy setup. In addition, hostname and IP address of the controller node should also be included. For example:
  
  ```
  export no_proxy=localhost,10.11.12.2,sdnlab-k01,10.11.12.1
  ```

- Differences in the `local.conf` file:
  
  - The service host is the controller, as well as other OpenStack servers, such as MySQL, Rabbit, Keystone, and Image. Therefore, they should be spelled out. Using the controller node example in the previous section, the service host and its IP address should be:
    
    ```
    SERVICE_HOST_NAME=sdnlab-k01
    SERVICE_HOST=10.11.12.1
    ```

  - The only OpenStack services required in compute nodes are messaging, nova compute, and neutron agent, so the `local.conf` might look like:
    
    ```
    disable_all_services
    enable_service rabbit
    enable_service n-cpu
    enable_service q-agt
    ```

  - The user has option to use `ovdk` or `openvswitch` for neutron agent:
    
    ```
    Q_AGENT=ovdk
    ```
    
    or
    
    ```
    Q_AGENT=openvswitch
    ```

  **Note:** For `openvswitch`, the user can specify regular or accelerated openvswitch (accelerated OVS). If accelerated OVS is use, the following setup should be added:

  ```
  OVS_DATAPATH_TYPE=netdev
  ```

  **Note:** If both are specified in the same `local.conf` file, the later one overwrites the previous one.

  - For the OVDK and accelerated OVS huge pages setting, specify number of huge pages to be allocated and mounting point (default is `/mnt/huge/`).
    
    ```
    OVDK_NUM_HUGEPAGES=8192
    ```
    
    or
    
    ```
    OVS_NUM_HUGEPAGES=8192
    ```

  - For this version, we use a specific OVDK version from its git repository, specify the following in the `local.conf` file if OVDK agent is used:
    
    ```
    OVDK_GIT_TAG=aca86d5b3663217c7d2e9e2a3d2d1037f21afbb6
    ```

  - Binding the physical port to the bridge is through the following line in `local.conf`. For example, to bind port p1p1 to bridge br-p1p1, use:
    
    ```
    OVS_PHYSICAL_BRIDGE=br-p1p1
    ```
— A sample `local.conf` file for compute node with ovdk agent follows:

```plaintext
# Compute node
[[local|localrc]]

FORCE=yes
MULTI_HOST=True

HOST_NAME=$(hostname)
HOST_IP=10.11.12.2
HOST_IP_IFACE=ens2f0
SERVICE_HOST_NAME=10.11.12.1
SERVICE_HOST=10.11.12.1

MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST

GLANCE_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST

ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
DATABASE_PASSWORD=password
SERVICE_TOKEN=no-token-password
HORIZON_PASSWORD=password
RABBIT_PASSWORD=password

disable_all_services

enable_service rabbit
enable_service n-cpu
enable_service q-agt

Q_AGENT=ovdk
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch
Q_ML2_PLUGIN_TYPE_DRIVERS=vlan
OVDK_NUM_HUGEPAGES=8192
OVDK_GIT_TAG=aca86d5b3663217c7d2e9e2a3d2d1037f21afbb6

DEST=/opt/stack
LOGFILE=$DEST/stack.sh.log
SCREEN_LOGDIR=$DEST/screen
SYSLOG=True
LOGDAYS=1

ENABLE_TENANT_VLANS=True
ML2_VLAN_RANGES=physnet1:1000:1010
PHYSICAL_NETWORK=physnet1
OVS_PHYSICAL_BRIDGE=br-p1p1

[[post-config|$NOVA_CONF]]
[DEFAULT]
firewall_driver=nova.virt.firewall.NoopFirewallDriver
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=$HOST_IP
```
A sample `local.conf` file for compute node with accelerated ovs agent follows.

```
# Compute node
# [[local|localrc]]

FORCE=yes
MULTI_HOST=True

HOST_NAME=$(hostname)
HOST_IP=10.11.12.2
HOST_IP_IFACE=ens2f0
SERVICE_HOST_NAME=sdnlab-k01
SERVICE_HOST=10.11.12.1

MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST

GLANCE_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST

ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
DATABASE_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=no-token-password
HORIZON_PASSWORD=password
RABBIT_PASSWORD=password

disable_all_services

enable_service rabbit
enable_service n-cpu
enable_service q-agt

Q_AGENT=openvswitch
Q_ML2PLUGIN_MECHANISM_DRIVERS=openvswitch
Q_ML2PLUGIN_TYPE_DRIVERS=vlan

OVS_NUM_HUGEPAGES=8192
OVS_DATAPATH_TYPE=netdev
OVS_GIT_TAG=d1279464ccfc6321075174f04b9df522b24cb674

DEST=/opt/stack
LOGFILE=$DEST/stack.sh.log
SCREEN_LOGDIR=$DEST/screen
SYSLOG=True
LOGDAYS=1

ENABLE_TENANT_VLANS=True
ML2_VLAN_RANGES=physnet1:1000:1010
PHYSICAL_NETWORK=physnet1
OVS_PHYSICAL_BRIDGE=br-p1p1

[[post-config|$NOVA_CONF]]
[DEFAULT]
firewall_driver=nova.virt.firewall.NoopFirewallDriver
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=$HOST_IP
```
5.3.1.3 vIPS

The vIPS used is Suricata, which should be installed as an rpm package as previously described. In order to configure it to run in inline mode (IPS) use the following:

1. Turn on IP forwarding.
   
   # sysctl -w net.ipv4.ip_forward=1

2. Mangle all traffic from one vPort to the other using a netfilter queue.
   
   # iptables -I FORWARD -i eth1 -o eth2 -j NFQUEUE
   # iptables -I FORWARD -i eth2 -o eth1 -j NFQUEUE

3. Have Suricata run in inline mode using the netfilter queue.
   
   # suricata -c /etc/suricata/suricata.yaml -q 0

4. Enable ARP proxying.
   
   # echo 1 > /proc/sys/net/ipv4/conf/eth1/proxy_arp
   # echo 1 > /proc/sys/net/ipv4/conf/eth2/proxy_arp

5.3.1.4 Network Configuration for non-vIPS Guests

1. Turn on IP forwarding.
   
   # sysctl -w net.ipv4.ip_forward=1

2. In the source, add the route to the sink.
   
   # route add -net 192.168.200.0/24 eth1

3. At the sink, add the route to the source.
   
   # route add -net 192.168.100.0/24 eth1
NOTE: This page intentionally left blank.
6.0 Testing the Controller Node

This section describes how to bring up the VMs in a compute node, connect them to the virtual network(s), verify the functionality.

Note: Currently, it is not possible to have more than one virtual network in a multi-compute node setup. Although, it is possible to have more than one virtual network in a single compute node setup.

6.1 Preparation with OpenStack

6.1.1 Deploying Virtual Machines

6.1.1.1 Default Settings

OpenStack comes with the following default settings:

- Tenant (Project): admin, demo
- Network:
  - Private network (virtual network): 10.0.0.0/24
  - Public network (external network): 172.24.4.0/24
- Image: cirros-0.3.1-x86_64
- Flavor: nano, micro, tiny, small, medium, large, xlarge

To deploy new instances (VMs) with different setups (such as a different VM image or network) users must create their own.

To access the OpenStack dashboard, use a web browser (like Firefox) and the controller's IP address (management network). For example:

http://10.11.12.1/

Login information is defined in the local.conf file. In the examples that follow, password is the password for both admin and demo users.
6.1.1.2 Customer Settings

The following example describes how to use a customer VM image and create a host aggregate and an available zone.

1. Use admin user to login from the OpenStack dashboard.

   http://10.11.12.1/dashboard

2. Upon a successful login, create VM image(s). Click the Images tab under the System Panel in the left pane, and click the Create Image tab in the upper-right corner.

3. In the Create An Image window, enter image name and description, then select the image source (the source should be accessible by OpenStack) and format from the respective drop-down boxes. Click Create Image at the bottom right to loads the image file to the controller host.
4. Create the available zone and host aggregate. Click **Host Aggregates** under **System Panel** on the left pane, then click **Create Host Aggregate** in the upper-right corner.

5. In the **Create Host Aggregate** window, enter names of the aggregate and availability zone.

6. Click the **Hosts within aggregate** tab (all available hosts are listed). Select host(s) to add into the aggregate.

7. Click **Create Host Aggregate** to finish.
6.1.1.3 Example — VM Deployment

The following example describes how to use a customer VM image and aggregate to launch a VM in an OpenStack environment.

1. Login as demo user.

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

3. Click the Networking tab, then select one or more networks for the instance.

4. Click Launch to finish.
5. The new VM should be up and running a few minutes. Click the new VM’s name in the list, then click **Console** in the top menu to access the VM.

6.1.1.4 Example — Creating an Additional Network

The following example describes how to create a new network in an OpenStack environment and how to apply to a VM to enable the VM to have multiple network interfaces.

1. Use demo user to login.
2. Click the **Network / Networks** tab in the left pane.
3. Click **Create Network** in the upper-right corner.
4. In the **Create Network** window, click the **Network** tab, then enter the network name. Click the **Subnet** tab and enter a subnet name, network address, and gateway. Click **Next** to continue. Note that users can ignore DNS and router setup and complete creating the network.

5. The two networks are now available. Assign both networks when creating an instance.

6. The instance has two network interfaces, belonging to two different networks.
6.1.1.5 Local vIPS

Configuration
1. OpenStack brings up the VMs and connects them to the vSwitch.
2. IP addresses of the VMs get configured using the DHCP server. VM1 belongs to one subnet and VM3 to a different one. VM2 has ports on both subnets.
3. Flows get programmed to the vSwitch by the OpenDaylight controller (Section 6.2).

Data Path (Numbers Matching Red Circles)
1. VM1 sends a flow to VM3 through the vSwitch.
2. The vSwitch forwards the flow to the first vPort of VM2 (active IPS).
3. The IPS receives the flow, inspects it and (if not malicious) sends it out through its second vPort.
4. The vSwitch forwards it to VM3.
6.1.1.6 Remote vIPS

Configuration
1. OpenStack brings up the VMs and connects them to the vSwitch.
2. The IP addresses of the VMs get configured using the DHCP server.

Data Path (Numbers Matching Red Circles)
1. VM1 sends a flow to VM3 through the vSwitch inside compute node 1.
2. The vSwitch forwards the flow out of the first 82599 port to the first 82599 port of compute node 2.
3. The vSwitch of compute node 2 forwards the flow to the first port of the vHost, where the traffic gets consumed by VM1.
4. The IPS receives the flow, inspects it, and (provided it is not malicious) sends it out through its second port of the vHost into the vSwitch of compute node 2.
5. The vSwitch forwards the flow out of the second 82599 port of compute node 2 into the second port of the 82599 in compute node 1.
6. The vSwitch of compute node 1 forwards the flow into the port of the vHost of VM3 where the flow gets terminated.

Figure 6-2 Remote iVPS
6.2 Using OpenDaylight

The previous section describes how to bring up several example configurations using OpenStack. This section describes how to use OpenDaylight. In this release (SRT 1.1), OpenDaylight is not integrated with OpenStack and the Intel® DPDK Accelerated vSwitch. As a result, this section points out some of the issues that result and how to use OpenDaylight independently to experiment with the local vIPS configuration.

6.2.1 Preparing OpenDaylight to Connect with the Compute Host

When using OpenStack to create networks and instantiate VMs (Section 6.1), OpenFlow rules are programmed into the vSwitch bridge instances running on the controller and compute hosts. Since OpenDaylight is not integrated as part of this process and when OpenDaylight is configured to be the OVSDB manager and OpenFlow controller of the bridge instances, the existing flow rules are removed. Furthermore, OpenDaylight has some built-in default network functionality that programs in new rules as the VMs begin to interact with the network. However, the resulting OpenDaylight rules are not the same as the OpenStack rules. For example, VLANs that OpenStack uses to partition subnets are not used by OpenDaylight.

For the purposes of this solutions guide, the goal is to use OpenDaylight manually, so the first step is to disable the basic networking services (bundles) in OpenDaylight (see Figure 6-3 on page 44).

1. Telnet into the OpenDaylight OSGi console.
2. Find the simpleforwarding and l2agent bundle ID’s using the `ss` command
3. Use the stop command to stop these bundles.
4. Check that the bundles are in the resolved state.
5. Exit from the telnet session.

**Note:** This process needs to be performed each time the controller host running OpenDaylight is rebooted, or each time the OpenDaylight service restarts.

**Note:** Section 5.2.2 describes why and how to disable the OVSDB neutron bundle by removing the bundles .jar file from the OpenDaylight plug-in folder. Alternatively, the OVSDB neutron bundle can be disabled via the OSGi console as described for the simpleforwarding and l2agent bundles.

6.2.2 Connecting OpenDaylight to the Compute Host

Once prepared, OpenDaylight sets itself as the OVSDB manager of the compute node. There are a number of ways to do this, but one simple method is to log into the compute host and execute the following command:

```
# ovs-vsctl set-manager tcp:<IP address of OpenDaylight controller>:6640
```

For example, if the IP address of the controller host running the OpenDaylight controller is 10.11.12.1, then:

```
# ovs-vsctl set-manager tcp:10.11.12.1:6640
```
Once performed, OpenDaylight connects to the compute host as the OVSDB manager and automatically configures itself to be the OpenFlow controller for the vSwitch bridge instances. The OVSDB configuration should show that the controller host is active as the manager and controller for each bridge. Note that `isConnected: true` appears for the manager setting and the controller setting of each bridge instance (see Figure 6-4 on page 45).

![Figure 6-3 Disabling the simpleforwarding and l2agent Bundles in the OSGi Console](image-url)
Once OpenDaylight is connected as the manager, note that the OpenFlow rules for the bridge instances are removed, as shown in Figure 6-5.

Figure 6-4 Set the OVS Manager

Figure 6-5 Display OpenFlow Rules After OpenDaylight is Configured as Manager
6.2.3 Configuring OpenFlow Rules Using OpenDaylight

Using the local vIPS configuration from Section 6.1.1.5, this section describes how to use OpenDaylight to configure some OpenFlow rules. When interacting with OpenDaylight to configure OpenFlow rules, two sets of information are required:

- The vSwitch bridge where the VMs are connected.
- Bridge ports associated with the VMs.

6.2.3.1 Identifying the Bridge Instance in OpenDaylight

OpenStack sets up at least two bridges on the compute host. One is connected to the physical interface and the other named `br-int` (where the VMs are attached). In the setup used as an example for this section, the two bridges are named `br-p786p1` and `br-int`. These appear in the OpenDaylight web GUI as nodes (see Figure 6-6 for an example). Figure 6-6 shows how to determine which node corresponds to `br-p786p1` and `br-int`.

![Figure 6-6 OpenDaylight Connected to the Compute Host](image)

The node with more ports is `br-int` because it has several VMs connected to it. The following command queries OVSDB and shows the bridge names and associated datapath_ids.

```
sudo ovs-vsctl list bridge | grep -E 'name|datapath_id'
```

Figure 6-6 shows the node IDs that uses the datapath_id to build the node name. The command shown in Figure 6-7 provides the correlation of datapath_id and bridge name.

![Figure 6-7 Query Bridge Name and datapath_id](image)
6.2.3.2 Identifying br-int Ports in OpenDaylight

With the OpenDaylight br-int node on the compute host identified, the next step is to determine which port on br-int is connected to which VM. OpenStack creates network interfaces for the VMs with names comprised of the concatenation of usv and the first 11 characters of the OpenStack neutron port object's UUID string. Note that the usv string appears to vary depending on the type of port, but seems to be used for the vHost based VMs used in this solutions guide. OpenDaylight uses these network interface names as port names.

The following command uses the nova list and neutron port-list commands along with awk to print out a useful list of the network information associated with each VM. Before running the nova and neutron commands, the openrc script in the devstack directory should be executed as follows:

```
[stack@sdnlab-k06 devstack]$ . ./openrc admin demo
$ nova list --fields name,networks | \  
  awk 'BEGIN \  { FS = "|" } \  / [0-9a-f]{8}-/ \  { split($4, arr, ";"); \  for (i in arr) \  print gensub("","","",g", $3) \  gensub("=","","",arr[i]) \  } | \  awk '{print $0; cmd = "neutron port-list | grep " $3; system(cmd)" | \  awk '/^[^|].*/ \  { printf "VM: %s
network:	%s
IP:		%s
port name:	usv%.11s
MAC:		%s

" $1, $2, $3 }'  
```

The output of the command looks something like this:

```
VM: vm01
network: private
IP: 10.0.0.6
port name: usvb638e1e0-f8
MAC: fa:16:3e:6c:7d:2f

VM: vm02
network: new-net
IP: 11.0.0.4
port name: usv81e5744e-b3
MAC: fa:16:3e:77:a6:1b

VM: vm1-2
network: new-net
IP: 11.0.0.6
port name: usv98ea655f-ee
MAC: fa:16:3e:95:2f:9a

VM: vm1-2
network: private
IP: 10.0.0.8
port name: usv5ce0d3a2-44
MAC: fa:16:3e:54:a1:7e
```

This information enables an association of the ports as they appear in OpenDaylight (see Figure 6-8) with the specific VM network ports. The number in parenthesis following each port name is the associated OpenFlow port number (a field in OVSDB interface table).
6.2.3.3 Establishing VM-to-VM Communication

In the example system used here to show the local vIPS system, vm01 and vm02 are the source and sink VMs and vm1-2 is the vIPS VM that functions as a router between the two subnets as well as running the Suricata vIPS software.

Since OpenDaylight deleted the OpenFlow rules when it connected to the compute host as the controller, it can now be used to create some new OpenFlow rules to re-establish communication among VMs.

**Note:** While OpenDaylight does delete the OpenFlow rules when it connects to the host as a controller, it has been observed that OpenStack does attempt to restore the rules. When experimenting with OpenFlow rules using OpenDaylight as described in this section, it is recommended to set the OpenFlow priority field higher than any pre-existing rules to ensure that the new rules have the intended effect.

**Note:** The **Flows** tab of the OpenDaylight Web GUI only shows the flow rules added by OpenDaylight. However, by looking at the **Troubleshoot** tab, all of the flows of a bridge node can be viewed from the 'Flows' link associated with each bridge.

Using the OpenFlow port numbers from **Figure 6-8**, the following rules can be created.

**Table 6-1 OpenFlow Rules**

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>In port 6</td>
<td>Output to port 8 (connects 10.0.0.6 on vm01 to 10.0.0.8 on vm1-2).</td>
</tr>
<tr>
<td>In port 8</td>
<td>Output to port 6 (connects 10.0.0.8 on vm1-2 to 10.0.0.6 on vm01).</td>
</tr>
<tr>
<td>In port 7</td>
<td>Output to port 9 (connects 11.0.0.4 on vm02 to 11.0.0.6 on vm1-2).</td>
</tr>
<tr>
<td>In port 9</td>
<td>Output to port 7 (connects 11.0.0.6 on vm1-2 to 11.0.0.4 on vm02).</td>
</tr>
</tbody>
</table>
Refer to Appendix D.2 for more information about using the OpenDaylight GUI to create OpenFlow rules.

**Note:** For these tests, it might be necessary to disable the firewall and clear the `iptables` on the VMs.

### 6.2.3.4 Baseline vm01-to-vm02 via vm1-2

In this test, an iperf session is run from the iperf client on vm01 to iperf server on vm02. The traffic passes through the Suricata node running in vm1-2 based on the OpenFlow rules configured in the previous section. A sample output of the iperf session is shown in Figure 6-9.

![Figure 6-9 iperf from vm01-to-vm02 via vm1-2](image)

### 6.2.3.5 Bypassing vm1-2

In this section two OpenFlow rules are added to bypass vm1-2 to enable vm01 to communicate directly to vm02. Since vm01 and vm02 are on two different subnets, the rules modify the source and destination MAC addresses to make the packet appear that it is receiving the VM that arrived from the VM1-2 interface on it's own subnet.

Table 6-2 presents an outline of the flow details that need to be entered via the OpenDaylight GUI:
Note: The order in which the actions were added to the rule was seen to make a difference. Adding the output port action last, after modifying the MAC addresses, worked best.

Note: These new rules are set with a priority of 501 to make them higher in priority than the previous 4 rules that have a priority of 500 (the default priority used by the OpenDaylight GUI).

After these two rules are installed, another iperf session is run. The iperf session starts initially with the traffic flowing through vm1-2. Then, using the Flow Entries screen in the OpenDaylight GUI, the two bypass flows are installed (by pressing the Install Flow button for each flow). After a few time periods pass, the two bypass flows are uninstalled (by pressing the Uninstall Flow button for each flow).

<table>
<thead>
<tr>
<th>Table 6-2  Flow Details for Bypassing vm1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Match Settings</strong></td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>In port 6</td>
</tr>
<tr>
<td>Priority = 501 (make higher priority than other rules)</td>
</tr>
<tr>
<td>Ethertype = 0x800 (IP packets)</td>
</tr>
<tr>
<td>Src IP = 10.0.0.6</td>
</tr>
<tr>
<td>Dst IP = 11.0.0.4</td>
</tr>
</tbody>
</table>

| In port 7         | Modify Src MAC to fa:16:3e:54:a1:7e (make packet appear to come from 10.0.0.8 on vm1-2) |
| Priority = 501 (make higher priority than other rules) | Modify Dst MAC to fa:16:3e:6c:7d:2f correct dest MAC to match 10.0.0.6 on vm01) |
| Ethertype = 0x800 (IP packets) | Output to port 6  (connects 11.0.0.4 on vm02 to 10.0.0.6 on vm01) |
| Src IP = 11.0.0.4 |             |
| Dst IP = 10.0.0.6 |             |

**Figure 6-10 iperf from vm01-to-vm02 Bypassing vm1-2**

Note that the overall throughput is significantly higher in the middle segments of the iperf session than when the rules bypassing vm1-2 are installed.
6.2.3.6 Bypassing vm1-2 for Specific Flows

In this example, iperf is run with two threads and the bypass flow rules add matches for the TCP protocol and TCP port numbers of one of the flows to demonstrate one of the iperf flows passing through vm1-2 while the other iperf flow bypasses vm1-2.

Table 6-3 presents an outline of the flow details that need to be entered via the OpenDaylight GUI. The rules are the same as in the previous bypass example with the addition of the match settings in bold:

Table 6-3 Flow Details for Bypassing vm1-2 for Specific Flows

<table>
<thead>
<tr>
<th>Match Settings</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>In port 6</td>
<td>Modify Src MAC to fa:16:3e:95:2f:9a (make packet appear to come from 11.0.0.6 on vm1-2)</td>
</tr>
<tr>
<td>Priority = 501 (make higher priority than other rules)</td>
<td>Modify Dst MAC to fa:16:3e:77:a6:1b (correct dest MAC to match 11.0.0.4 on vm02)</td>
</tr>
<tr>
<td>Ethertype = 0x800 (IP packets)</td>
<td>Ouput to port 7 (connects 10.0.0.6 on vm01 to 11.0.0.4 on vm02)</td>
</tr>
<tr>
<td>Src IP = 10.0.0.6</td>
<td></td>
</tr>
<tr>
<td>Dst IP = 11.0.0.4</td>
<td></td>
</tr>
<tr>
<td>Layer 4 Protocol = 6 (TCP protocol)</td>
<td></td>
</tr>
<tr>
<td>Source Port = 43112 (TCP port of one of the flows)</td>
<td></td>
</tr>
<tr>
<td>In port 7</td>
<td>Modify Src MAC to fa:16:3e:54:a1:7e (make packet appear to come from 10.0.0.6 on vm01)</td>
</tr>
<tr>
<td>Priority = 501 (make higher priority than other rules)</td>
<td>Modify Dst MAC to fa:16:3e:6c:7d:2f (correct dest MAC to match 10.0.0.6 on vm01)</td>
</tr>
<tr>
<td>Ethertype = 0x800 (IP packets)</td>
<td>Ouput to port 6 (connects 11.0.0.6 on vm02 to 10.0.0.6 on vm01)</td>
</tr>
<tr>
<td>Src IP = 11.0.0.4</td>
<td></td>
</tr>
<tr>
<td>Dst IP = 10.0.0.6</td>
<td></td>
</tr>
<tr>
<td>Layer 4 Protocol = 6 (TCP protocol)</td>
<td></td>
</tr>
<tr>
<td>Destination Port = 43112 (TCP port of one flow)</td>
<td></td>
</tr>
</tbody>
</table>

Note: When modifying the bypass flow rules in the OpenDaylight GUI to add the Layer 4 match criteria, it was found that while the rule could be saved, it was not possible to install it (installing would fail). By logging out of the web GUI and then logging back in, it was then possible to install the modified rules.

Figure 6-11 shows the output of the iperf run with two threads. The bypass flow rules for the iperf thread using TCP port 43112 are installed after a couple intervals and then uninstalled after a couple more. The results show that the iperf flow on TCP port 43112 has much higher throughput than the other iperf flow while the bypass rules are installed.
6.2.3.7 Summary of OpenFlow Rules Created

Figure 6-12 shows a list of the OpenFlow rules created for br-int on the compute host via the OpenDaylight GUI (the output has been edited and notated for clarity).
7.0 Testing Compute Node Performance

Network throughput performance is verified using the L3 forwarding DPDK application in the VM. The VMs are created using the userspace vHost and IVSHMEM I/O virtualization method. To contrast the performance of the DPDK L3 forwarding test case in the userspace vHost and IVSHMEM type VM, the Linux kernel forwarding performance in VM that uses Open vSwitch with the standard vHost I/O virtualization method is included.

The DPDK L3 forwarding on host and port-port switching performance using DPDK vSwitch (Section 7.1 and Section 7.2) serve as verification tests after installing compute node using DevStack. All the performance tests have been performed by manually configuring the DPDK vSwitch bridges and ports, and adding flow to the DPDK vSwitch flowtable for forwarding tests on:

- Host
- VMs using:
  - Userspace vHost
  - IVSHMEM

Refer to appropriate sections in Appendix B.3 for detailed instructions.

7.1 Intel® DPDK and Linux Kernel L3 Forwarding on the Host

Figure 7-1 Testing L3 Forwarding
Configuration
1. No VMs are configured and no vSwitch is started or used.
2. DPDK-based port forwarding sample application (l3fwd) or Linux stack for port-to-port network throughput tests.

Data Path (Numbers Matching Red Circles)
1. The packet generator creates flows based on RFC 2544.
2. The DPDK L3 forwarding application or Linux stack forwards the traffic from the 82599's first physical port to the second.
3. The traffic flows back to the packet generator to complete the RFC2544 throughput test.

7.1.1 L3 Forwarding Performance Characterization
By comparing the L3 forwarding results using the DPDK and non-DPDK Linux kernel shows that DPDK gives a greater performance improvement for smaller size packets. For 64-byte Ethernet frames, DPDK gives 14 times performance improvement over the Linux stack.

![L3 Forwarding Performance Comparing Standard Linux Stack and DPDK](image)

Figure 7-2 L3 Forwarding Performance Comparing Standard Linux Stack and DPDK

---

1. Performance using RFC2544 zero loss methodology. Refer to Appendix A for more details.
7.2 Port-Port Forwarding on Host Using Intel® DPDK Accelerated Open vSwitch and Open vSwitch

Figure 7-3 Testing Port-to-Port Forwarding at the vSwitch

Configuration
1. No VMs are configured.
2. DPDK vSwitch / OVS-based, port-to-port network throughput tests

Data Path (Numbers Matching Red Circles)
1. The packet generator creates flows based on RFC 2544.
2. The DPDK vSwitch / OVS network stack forwards the traffic from the 82599’s first physical port to the second.
3. The traffic flows back to the packet generator.
7.2.1 Port-port vSwitch Performance Characterization

As shown Figure 7-4, the network throughput performance to switch packets on host is best when DPDK vSwitch is used followed by DPD Accelerated vSwitch and finally Open vSwitch.

Figure 7-4 Port-to-Port vSwitch Performance

7.3 L3 Forwarding in the VM Using Open vSwitch (OVS) and DPDK vSwitch

Figure 7-5 Testing L3 Forwarding in the VM Configuration

1. Performance using RFC2544 zero loss methodology. Refer to Appendix A for more details.
Configuration (Done Manually)
1. One VM gets brought up and connected to the vSwitch.
2. IP addresses 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.

7.3.1 Single VM L3 Forwarding Performance Characterization

As shown in Figure 7-6, performance greatly improved when using the DPDK accelerated vSwitch, especially with smaller-sized packets. For 64-byte packets, a 12 time performance increase was seen when using the DPDK vSwitch.

![L3 Forwarding in a VM](image)

**Figure 7-6 Single VM L3 Forwarding Performance**

1. Performance is based on RFC2544 with 0.01% allowable packet loss. Refer to Appendix A for more details.
7.4 L3 Forwarding in Two VMs Using Open vSwitch (OVS) and DPDK vSwitch with userspace vHost

Configuration (Done Manually)
1. Setup 2 VMs and connect to 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 VM #1.
3. VM #1 receives the flows and forwards them out to VM #2 through its second vPort.
4. VM #2 receives the flows and forwards it to the vSwitch through its second vPort.
5. The vSwitch forwards the flows back to the packet generator.
7.4.1 Two VMs L3 Forwarding Performance Characterization

As shown in Figure 7-8, another performance increase was seen for smaller-sized packets when comparing the DPDK accelerated vSwitch and Open vSwitch. The DPDK vSwitch is using userspace vHost while Open vSwitch is using Linux vHost for host/VM communication.

![L3 Forwarding in 2 VMs](image)

**Figure 7-8 Two VMs L3 Forwarding Performance**

7.4.2 Scalability Tests for Two VMs Using userspace vHost as vNICs

DPDK vSwitch offers multi-core performance for VM switching. This can be tested by allocating more cores to the DPDK vSwitch. Virtual ports are added to cores in a round robin fashion. Therefore, the performance scales by adding cores to the DPDK vSwitch. The performance chart in Figure 7-9 shows the comparison of performance for three different coremasks for DPDK vSwitch.

---

1. Performance is based on RFC2544 with 0.01% allowable packet loss. Refer to Appendix A for more details.
Figure 7-9  Scalability Tests for Two VMs Using userspace vHost as vNICs

7.5  VM-VM IPSec Tunnel Performance Using Quick Assist Technology

An IPSec Tunnel is setup between two tunnel endpoints, one on each Virtual Machine, and each VM uses Quick Assist technology to offload encryption and decryption. The virtual network interface of the VMs is of the type standard virtio with vHost as offered by standard Open vSwitch.

Figure 7-10  VM-VM IPSec Tunnel Performance

---

1. Performance is based on RFC2544 with 0.01% allowable packet loss. Refer to Appendix A for more details.
As shown in the performance comparison in Figure 7-11, using QAT gives a more than twice improvement in throughput for 64-byte packets, and improves further increases for larger sized packets. Note that the throughput numbers in Figure 7-11 were obtained using Netperf test methodology.

Figure 7-11  VM-to-VM IPSec Tunnel Performance Using Quick Assist Technology

1. Performance is based on Netperf UDP stream test methodology. Refer to Appendix A for more details.
NOTE: This page intentionally left blank.
Appendix A  Test Methodology

A.1 Defining 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.

RFC2544 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 RFC2544, unless otherwise noted. Frame size refers to Ethernet frames ranging from smallest frames of 64 bytes to largest frames of 1518 bytes. For 64-byte frames, a line rate of 10 Gb/s translates to 14.88 million packets per second for unidirectional traffic.

A.2 Network Protocols

Port forwarding and L3 forwarding test cases use IPv4 packets and UDP datagrams.

A.3 RFC2544

In this solutions guide, DPDK L3 forwarding on the host and port-to-port L3 forwarding using a vSwitch represents the fastest rate at which the system under test can forward Ethernet frames without any packet loss (RFC 2544 zero loss test).

Performance tests using a VM, however, have a deemed minimal acceptable packet loss of 0.01% (RFC2544 0.01% loss test).

A.4 Netperf

Netperf (http://www.netperf.org/netperf/) is a benchmark that can be used to measure various aspects of networking performance, and is used here for IPSec Tunnel performance. The Netperf UDP stream test reports UDP payload throughput. A 64-byte Ethernet frame contains an 18 bytes UDP payload. Therefore, a line rate of 10 Gb/s results in a maximum theoretical UDP payload throughput of 2.143 Gb/s.
Appendix B Installing and Running DPDK vSwitch on the CLI

Note: This guide describes how to use DevStack (http://devstack.org/) to deploy compute and controller nodes. In this Appendix, information is also provided on how to manually set up compute nodes. The two processes are quite different and should not be mixed together. Manually building compute node allows for full optimization, and promotes familiarity with all aspects of the deployment. All performance results have been captured with the manual procedure.

B.1 Intel DPDK 1.7 with Fedora 20

B.1.1 Building a DPDK Target from Source

1. Get DPDK release 1.7 and unzip it.

   # cd /usr/src
   # unzip ~/R1.7/DPDK-1.7.zip

2. Install DPDK target environments.

   # cd /usr/src/DPDK-1.7
   # make install T=x86_64-ivshmem-linuxapp-gcc

   ... Build complete

B.1.2 Hugepage Configuration

Hugepages enables the Linux kernel to use multiple page-size capabilities of modern hardware architectures. Linux uses pages as the basic unit of memory to partition physical memory and access it using basic page unit. The default page size is 4096 bytes in the x86.

Transaction Lookaside Buffers (TLB) contain mappings of virtual memory to actual physical memory addresses. Using a large number of pages with the default page size stresses the TLB. Using bigger page sizes to address physical memory improves program performance by reducing the frequency of TLB updates.

Intel® DPDK vSwitch configuration requires four 1 GB hugepage per instance and at least four more to run a VM using qemu. Update the /boot/grub2/grub.cfg with the following parameter and reboot the system for it to take effect.
Verify that you booted with huge pages support enabled by doing:

```
# dmesg | grep command
[ 0.000000] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-3.11.10-301.fc20.x86_64
default_hugepagesz=1G hugepagesz=1G hugepages=32 root=UUID=978575a5-45f3-4675-9e4b-
e17f3fd0a32
ro vconsole.font=latarcyrheb-sun16 rhgb quiet LANG=en_US.UTF-8
```

For manual setup only (non DevStack), once Hugepages are allocated on the host, a file mount point directory needs to be created in order to mount the hugepage file system, as follows:

```
# mkdir -p /dev/hugepages
# mount -t hugetlbfs nodev /dev/hugepages
```

## B.1.3 Loading DPDK igb_uio and Binding the Drivers to 10 GbE Interfaces

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

Bind/Unbind to/from the igb_uio module.

```
# cd ..
# ./tools/dpdk_nic_bind.py --status
Network devices using IGB_UIO driver

```
```
Network devices using kernel driver
```

```
0000:01:00.0 '82599ES 10-Gigabit SFI/SFP+ Network Connection' if=p1p1 drv=ixgbe unused=igb_uio
0000:01:00.1 '82599ES 10-Gigabit SFI/SFP+ Network Connection' if=p1p2 drv=ixgbe unused=igb_uio
0000:08:00.0 'I350 Gigabit Network Connection' if=em0 drv=igb unused=igb_uio *Active*
0000:08:00.1 'I350 Gigabit Network Connection' if=em0f01 drv=igb unused=igb_uio
0000:08:00.2 'I350 Gigabit Network Connection' if=em0f02 drv=igb unused=igb_uio
0000:08:00.3 'I350 Gigabit Network Connection' if=em0f03 drv=igb unused=igb_uio
```

Other network devices

```
<none>
```

To bind devices p1p1 and p1p2, (01:00.0 and 01:00.1), to the igb_uio driver.

```
# ./tools/dpdk_nic_bind.py --bind=igb_uio 01:00.0
# ./tools/dpdk_nic_bind.py --bind=igb_uio 01:00.1
# ./tools/dpdk_nic_bind.py --status
```

```
Network devices using IGB_UIO driver
```

```
0000:01:00.0 '82599ES 10-Gigabit SFI/SFP+ Network Connection' drv=igb_uio unused=
0000:01:00.1 '82599ES 10-Gigabit SFI/SFP+ Network Connection' drv=igb_uio unused=
```

Network devices using kernel driver

```
```
```
0000:08:00.0 'I350 Gigabit Network Connection' if=em0 drv=igb unused=igb_uio *Active*
0000:08:00.1 'I350 Gigabit Network Connection' if=em0f01 drv=igb unused=igb_uio
0000:08:00.2 'I350 Gigabit Network Connection' if=em0f02 drv=igb unused=igb_uio
0000:08:00.3 'I350 Gigabit Network Connection' if=em0f03 drv=igb unused=igb_uio
```

Other network devices

```
<none>
```
B.2 Guest Operating System Installation and Configuration

B.2.1 Creating VM Disk and Starting OS Install

Create an empty qcow2 image file using `qemu-img create` command.

```
# qemu-img create -f qcow2 /var/lib/libvirt/images/fedora20.qcow2 16G
Formatting '/var/lib/libvirt/images/fedora20.qcow2', fmt=qcow2 size=17179869184
encryption=off
cluster_size=65536 lazy_refcounts=off
Start the VM for installation of Fedora 20 guest:
# /usr/bin/qemu-system-x86_64 \
-boot d -m 4096M -smp 2 -vnc :2 \
-hda /var/lib/libvirt/images/fedora20.qcow2\n-cdrom <Install_Disk_location_for_Fedora-20-x86_64-DVD.iso>
```

B.3 Intel® DPDK vSwitch 1.0 with Fedora 20

B.3.1 Building DPDK vSwitch on Host

Untar the DPDK vSwitch code to the build directory.

```
# cd /usr/src/
# tar -xvzf dpdk-ovs-1.1.0.tar.gz
# cd dpdk-ovs-1.1.0/openvswitch
# ./boot.sh
# ./configure RTE_SDK=/usr/src/DPDK-1.7 --disable-ssl
# make
```

B.3.2 Configuring Kernel Variables

```
# 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
```

B.3.3 Building QEMU for Intel® DPDK vSwitch

```
# cd <Intel DPDK vSwitch_INSTALL_DIR>/qemu
# ./configure --enable-kvm --target-list=x86_64-softmmu --disable-pie
# make
```
B.3.4 PHY-PHY Throughput Test on Host Using DPDK vSwitch

1. Start with a clean system.

   # cd <Intel DPDK vSwitch_INSTALL_DIR>
   # pkill -9 ovs
   # rm -rf /usr/local/var/run/openvswitch/
   # rm -rf /usr/local/etc/openvswitch/
   # mkdir -p /usr/local/var/run/openvswitch/
   # mkdir -p /usr/local/etc/openvswitch/
   # rm -f /tmp/conf.db
   # rmmod vhost-net
   # rm -rf /dev/vhost-net
   # rmmod ixgbe
   # rmmod uio
   # rmmod igb_uio
   # umount /sys/fs/cgroup/hugetlb
   # umount /dev/hugepages
   # umount /mnt/huge

2. Mount hugepage filesystem on host.

   # mount -t hugetlbfs nodev /dev/hugepages
   # mount|grep huge

3. Install the DPDK kernel modules:

   # cd <Intel DPDK vSwitch_INSTALL_DIR>
   # modprobe uio
   # insmod <DPDK_INSTALL_DIR>/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko

4. Bind the physical interfaces to igb_uio driver. For example, if the PCI addresses on the setup are 08:00.0 and 08:00.1, the following command shows how to bind them to the DPDK driver. Replace the PCI addresses according to the setup.

   # ./dpdk*/tools/dpdk_nic_bind.py --status
   # ./dpdk*/tools/dpdk_nic_bind.py --bind=igb_uio 08:00.0
   # ./dpdk*/tools/dpdk_nic_bind.py --bind=igb_uio 08:00.1

5. Create an Intel® DPDK vSwitch database file.

   # ./ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db <OVS_INSTALL_DIR>/openvswitch/vswitchd/vswitch.ovsschema

6. Run the ovsdb-server.

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

7. Create Intel DPDK® vSwitch bridge, add physical interfaces (type: dpdkphy) to the bridge:

   # ./utilities/ovs-vsctl --no-wait add-br br0 -- set Bridge br0 datapath_type=dpdk
   # ./utilities/ovs-vsctl --no-wait add-port br0 port1 -- set Interface port1 type=dpdkphy ofport_request=1 option:port=0
   # ./utilities/ovs-vsctl --no-wait add-port br0 port2 -- set Interface port2 type=dpdkphy ofport_request=1 option:port=1
The output should be:

```
00000000-0000-0000-0000-000000000000
Bridge "br0"
  Port "br0"
    Interface "br0"
      type: internal
  Port "port1"
    Interface "port1"
      type: dpdkphy
        options: {port="0"}
  Port "port2"
    Interface "port2"
      type: dpdkphy
        options: {port="1"}
```

8. Start the Intel® DPDK vSwitch for the PHY-PHY throughput test.

```
# ./datapath/dpdk/ovs-dpdk -c 0x0F -n 4 --proc-type primary --socket-mem 4096 --
  -p 0x03 --stats_core 0 --stats_int 5
```

Exit the process using **CTRL-C**. Copy the valid address on the system from the stdout output of the above command. For example, EAL: virtual area found at 0x7F1740000000 (size = 0x80000000) and use it for the base-virtaddr parameter of the following command:

```
# ./datapath/dpdk/ovs-dpdk -c 0x0F -n 4 --proc-type primary --base-virtaddr=0x7f9a40000000 --socket-mem 4096 -- -p 0x03 --stats_core 0 --stats_int 5
```

9. Run the vswitchd daemon.

```
# ./vswitchd/ovs-vswitchd -c 0x100 --proc-type=secondary -- --pidfile=/tmp/vswitchd.pid
```

10. Using the **ovs-ofctl** utility, add flows for bidirectional test. The following example shows how to add flows when the source and destination IPs are 1.1.1.1 and 6.6.6.2 for a bidirectional use case.

```
# ./utilities/ovs-ofctl del-flows br0
# ./utilities/ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_src=1.1.1.1,
  nw_dst=6.6.6.2,idle_timeout=0,action=output:2
# ./utilities/ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_src=6.6.6.2,nw_dst=1.1.1.1,
  idle_timeout=0,action=output:1
# ./Utilities/ovs-ofctl dump-flows br0
# ./utilities/ovs-vstctl --no-wait -- set Open_vSwitch.other_config:n-handler-threads=1
```
B.3.5 Userspace vHost Inter-VM Communication Method

B.3.5.1 Host Configuration for Running a VM Using userspace vHost

1. Start with a clean system.

   # cd <Intel DPDK vSwitch_INSTALL_DIR>
   # pkill -9 ovs
   # rm -rf /usr/local/var/run/openvswitch/
   # rm -rf /usr/local/etc/openvswitch/
   # mkdir -p /usr/local/var/run/openvswitch/
   # mkdir -p /usr/local/etc/openvswitch/
   # rm -f /tmp/conf.db
   # rmmod vhost-net
   # rm -rf /dev/vhost-net
   # rmmod ixgbe
   # rmmod uio
   # rmmod igb_uio
   # umount /sys/fs/cgroup/hugetlb
   # umount /dev/hugepages
   # umount /mnt/huge

2. Mount hugepage filesystem on host.

   # mount -t hugetlbfs nodev /dev/hugepages
   # mount|grep huge

3. Install the DPDK, cuse and eventfd kernel modules.

   # cd <Intel DPDK vSwitch_INSTALL_DIR>
   # modprobe cuse
   # insmod ./openvswitch/datapath/dpdk/fd_link/fd_link.ko
   # modprobe uio
   # insmod <DPDK_INSTALL_DIR>/x86_64-ixshmem-linuxapp-gcc/kmod/igb_uio.ko

4. Bind/Unbind to/from the igb_uio module.

   # <DPDK_INSTALL_DIR>/tools/dpdk_nic_bind.py --status Network devices using IGB_UIO driver
   =====================================
   <none>
   Network devices using kernel driver
   0000:01:00.0 '82599ES 10-Gigabit SFP/SFP+ Network Connection' if=p1p1 drv=ixgbe unused=igb_uio
   0000:01:00.1 '82599ES 10-Gigabit SFP/SFP+ Network Connection' if=p1p2 drv=ixgbe unused=igb_uio
   0000:08:00.0 '1350 Gigabit Network Connection' if=em0 drv=igb unused=igb_uio *Active*
   0000:08:00.1 '1350 Gigabit Network Connection' if=enp8s0f1 drv=igb unused=igb_uio
   0000:08:00.2 '1350 Gigabit Network Connection' if=enp8s0f2 drv=igb unused=igb_uio
   0000:08:00.3 '1350 Gigabit Network Connection' if=enp8s0f3 drv=igb unused=igb_uio Other network devices
   ======================<none>

   To bind devices p1p1 and p1p2, (01:00.0 and 01:00.1), to the igb_uio driver.

   # <DPDK_INSTALL_DIR>/tools/dpdk_nic_bind.py --bind=igb_uio 01:00.0
   # <DPDK_INSTALL_DIR>/tools/dpdk_nic_bind.py --bind=igb_uio 01:00.1
   # <DPDK_INSTALL_DIR>/tools/dpdk_nic_bind.py --status Network devices using IG8_UIO driver
   ===========================================================================
   0000:01:00.0 '82599ES 10-Gigabit SFP/SFP+ Network Connection' drv=igb_uio unused=
   0000:01:00.1 '82599ES 10-Gigabit SFP/SFP+ Network Connection' drv=igb_uio unused= Network devices using igb_uio kernel driver
   ===========================================================================
   0000:08:00.0 '1350 Gigabit Network Connection' if=en0 drv=igb unused=igb_uio *Active*
   0000:08:00.1 '1350 Gigabit Network Connection' if=enp8s0f1 drv=igb unused=igb_uio
   0000:08:00.2 '1350 Gigabit Network Connection' if=enp8s0f2 drv=igb unused=igb_uio
   0000:08:00.3 '1350 Gigabit Network Connection' if=enp8s0f3 drv=igb unused=igb_uio Other network devices
   ======================<none>

5. Create an Intel® DPDK vSwitch database file.

   # ./ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db <OVS_INSTALL_DIR>/openvswitch/vswitchd/vswitchd.ovsschema
6. Run the **ovsdb-server**.

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

7. Create Intel DPDK® vSwitch bridge, add physical interfaces (type: dpdkphy) and vhost interfaces (type: dpdkvhost) to the bridge.

   ```bash
   # ./utilities/ovs-vsctl --no-wait add-br br0 -- set Bridge br0 datapath_type=dpdk
   # ./utilities/ovs-vsctl --no-wait add-port br0 port1 -- set Interface port1 type=dpdkphy
   #    ofport_request=1 option:port=0
   # ./utilities/ovs-vsctl --no-wait add-port br0 port2 -- set Interface port2 type=dpdkphy
   #    ofport_request=2 option:port=1
   # ./utilities/ovs-vsctl --no-wait add-port br0 port3 -- set Interface port3 type=dpdkvhost
   #    ofport_request=3
   # ./utilities/ovs-vsctl --no-wait add-port br0 port4 -- set Interface port4 type=dpdkvhost
   #    ofport_request=4
   # ./utilities/ovs-vsctl show
   ``

The output should be:

```
00000000-0000-0000-0000-000000000000
Bridge "br0"
  Port "br0"
  Interface "br0"
    type: internal
Port "port16"
  Interface "port1"
    type: dpdkphy
    options: {port="1"}
Port "port17"
  Interface "port2"
    type: dpdkphy
    options: {port="2"}
Port "port80"
  Interface "port3"
    type: dpdkvhost
Port "port81"
  Interface "port4"
    type: dpdkvhost
```

8. Start the Intel® DPDK vSwitch for 1 VM with two userspace vHost interfaces.

   ```bash
   # ./datapath/dpdk/ovs-dpdk -c 0x0F -n 4 --proc-type primary --socket-mem 4096 --
   -p 0x03 --stats_core 0 --stats_int 5
   ``

Exit the process using **CTRL-C**. Copy the valid address on the system from the stdout output of the above command. For example, EAL: virtual area found at 0x7F1740000000 (size = 0x80000000). Use it for the base-virtaddr parameter of the following command:

   ```bash
   # ./datapath/dpdk/ovs-dpdk -c 0x0F -n 4 --proc-type primary --base-virtaddr=0x7f9a40000000 --socket-mem 4096 --
   -p 0x03 --stats_core 0 --stats_int 5
   ```

9. Run the vswitchd daemon.

   ```bash
   # ./vsswitchd/ovs-vswitchd -c 0x100 --proc-type=secondary --pidfile=/tmp/vswitchd.pid
   ```

10. Using the **ovs-ofctl** utility, add flows entries to switch packets from port 2 to port 4, port 3 to port 1 on ingress and egress path. The following example shows how to add flows when source and destination IPs are 1.1.1.1 and 6.6.6.2 for a bidirectional case.

    ```bash
    # ./utilities/ovs-ofctl del-flows br0
    # ./utilities/ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_src=1.1.1.1,nw_dst=6.6.6.2,
    idle_timeout=0,action=output:4
    # ./utilities/ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,nw_src=1.1.1.1,nw_dst=6.6.6.2,
    idle_timeout=0,action=output:1
    # ./Utilities/ovs-ofctl add-flow br0 in_port=4,dl_type=0x0800,nw_src=6.6.6.2,nw_dst=1.1.1.1,
    ```
idle_timeout=0,action=output:2
# ./Utilities/ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_src=6.6.6.2,nw_dst=1.1.1.1,
idle_timeout=0,action=output:3
# ./Utilities/ovs-ofctl dump-flows br0

11. Copy DPDK source to a shared directory to be passed to the VM.

    # rm -rf /tmp/qemu_share
    # mkdir -p /tmp/qemu_share
    # mkdir -p /tmp/qemu_share/DPDK
    # chmod 777 /tmp/qemu_share
    # cp -aL <DPDK_INSTALL_DIR>/* /tmp/qemu_share/DPDK

12. Start the vm using qemu command.

    # cd <Intel DPDK vSwitch_INSTALL_DIR>
    # taskset 0x30 ./qemu/x86_64-softmmu/qemu-system-x86_64-cpu host -boot c -hda
    <VM_IMAGE_LOCATION> -m 4096 -smp 2 --enable-kvm -name 'client 1' -nographic -vnc :12 -
    pidfile /tmp/vmpid -drive file=fat:rw:/tmp/qemu_share -monitor unix:/tmp
    vmpimonitor,server,nowait -net none -no-reboot -mem-path /dev/hugepages -mem-prealloc -netdev
    type=tap,id=net1,script=no,downscript=no,ifname=port80,host=on -device virtio-net-pci,
etdev=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=port81,host=on -device virtio-net-pci,
etdev=net2,mac=00:00:00:00:00:02,csum=off,gso=off,guest_tso4=off,
guest_tso6=off,guest_ecn=off

The VM can be accessed using a VNC client at the port mentioned in the qemu startup command, which
is port 12 in this instance.

B.3.5.2 Guest Configuration to Run DPDK vHost Sample Forwarding Application

1. Install DPDK on the VM using the source from the shared drive.

    # mkdir -p /mnt/vhost
    # mkdir -p /root/vhost
    # mount -o iocharset=utf8 /dev/sdb1 /mnt/vhost
    # cp -a /mnt/vhost/* /root/vhost
    # export RTE_SDK=/root/vhost/DPDK
    # export RTE_TARGET=x86_64-ivshmem-linuxapp-gcc
    # cd /root/vhost/DPDK
    # make uninstall
    # make install T=x86_64-ivshmem-linuxapp-gcc

2. Build the DPDK vHost application testpmd.

    # cd /root/vhost/DPDK/app/test-pmd
    # make clean
    # make

3. Run the testpmd application after installing the UIO drivers.

    # modprobe uio
    # echo 1280 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/
    nr_hugepages
    # insmod /root/vhost/DPDK/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko
    # /root/vhost/DPDK/tools/dpdknic_bind.py --bind igb_uio 0000:00:03.0 0000:00:04.0
    #./testpmd -c 0x3 -n 4 --socket-mem 128 -- --burst=64 -i

At the prompt enter the following to start the application:

    testpmd> set fwd mac_retry
testpmd> start
B.3.6 IVSHMEM Inter-VM Communication Method

B.3.6.1 Host Configuration for Running a VM Using ivshmem

1. Mount hugepage filesystem on host.
   
   # mount -t hugetlbfs nodev /dev/hugepages
   # mount|grep huge

2. Initialize the Open vSwitch database server.

   # mkdir -p /usr/local/etc/openvswitch
   # mkdir -p /usr/local/var/run/openvswitch
   # mkdir -p /usr/local/var/log/openvswitch
   # cd /usr/src/dpdk-ovs/openvswitch/
   # rm -f /usr/local/etc/openvswitch/conf.db
   # ./ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db vswitchd/
   vswitch.ovsschema
   # ./ovsdb/ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \\
   --remote=db:Open_vSwitch,Open_vSwitch,manager_options --pidfile --detach
   # ps -e|grep ovsdb-server
   24309 ?00:00:00 ovsdb-server
   # ./utilities/ovs-vsctl show
   6b60bd8d-9463-46b5-a173-a6f39670b955

3. Add a bridge to the switch.

   # ./utilities/ovs-vsctl -no-wait add-br br0 - set Bridge br0 datapath_type=dpdk

4. Add ports to the bridge.

   # ./utilities/ovs-vsctl --no-wait add-port br0 port1 -- set Interface port1
   type=dpdkphy ofport_request=1 option:port=0
   # ./utilities/ovs-vsctl --no-wait add-port br0 port2 -- set Interface port2
   type=dpdkphy ofport_request=2 option:port=1
   # ./utilities/ovs-vsctl --no-wait add-port br0 port3 -- set Interface port3
   type=dpdkclient ofport_request=3

5. Confirm the ports have been successfully added.

   # ./utilities/ovs-vsctl show

   Users should see something like this:

   00000000-0000-0000-0000-000000000000
   Bridge "br0"
   Port "br0"
   Interface "br0"
   type: internal
   Port "port1"
   Interface "port1"
   type: dpdkphy
   options: {port="0"}
   Port "port2"
   Interface "port2"
   type: dpdkphy
   options: {port="1"}
   Port "port3"
   Interface "port3"
   type: dpdkclient

----------------------------------------------

```
# ./datapath/dpdk/ovs-dpdk -c 0x0F -n 4 --proc-type primary -- -p 0x03
--stats_core 0 --stats_int 5
```

Running the previous command should generate an output like the following:

```
EAL: Detected lcore 0 as core 0 on socket 0
EAL: Detected lcore 1 as core 1 on socket 0
EAL: Detected lcore 2 as core 2 on socket 0
EAL: Detected lcore 3 as core 3 on socket 0
EAL: Searching for IVSHMEM devices...
EAL: No IVSHMEM configuration found!
EAL: Setting up memory...
EAL: Ask a virtual area of 0x6000000000 bytes
EAL: Virtual area found at 0x7ff940000000 (size = 0x6000000000)
EAL: Ask a virtual area of 0x6000000000 bytes
EAL: Virtual area found at 0x7ff300000000 (size = 0x6000000000)
EAL: Requesting 24 pages of size 1024MB from socket 0
EAL: TSC frequency is ~2793266 KHz
EAL: Master core 0 is ready (tid=f7fe1980)
EAL: Core 1 is ready (tid=f77a3700)
EAL: Core 2 is ready (tid=f6fa2700)
EAL: Core 3 is ready (tid=f67a1700)
EAL: PCI device 0000:04:00.0 on NUMA socket 0
EAL: probe driver: 8086:1521 rte_igb_pmd
EAL: 0000:04:00.0 not managed by UIO driver, skipping
EAL: PCI device 0000:04:00.1 on NUMA socket 0
EAL: probe driver: 8086:1521 rte_igb_pmd
EAL: 0000:04:00.1 not managed by UIO driver, skipping
EAL: PCI device 0000:04:00.2 on NUMA socket 0
EAL: probe driver: 8086:1521 rte_igb_pmd
EAL: 0000:04:00.2 not managed by UIO driver, skipping
EAL: PCI device 0000:04:00.3 on NUMA socket 0
EAL: probe driver: 8086:1521 rte_igb_pmd
EAL: 0000:04:00.3 not managed by UIO driver, skipping
EAL: PCI device 0000:08:00.0 on NUMA socket 0
EAL: probe driver: 8086:10fb rte_ixbg_pmd
EAL: PCI memory mapped at 0x7ffff5f21000
EAL: PCI memory mapped at 0x7ffff7fec000
EAL: PCI device 0000:08:00.1 on NUMA socket 0
EAL: probe driver: 8086:10fb rte_ixbg_pmd
EAL: PCI memory mapped at 0x7ffffff5f21000
EAL: PCI memory mapped at 0x7ffffff5f21000
```

Look for similar line as previously highlighted in red text. Use the address to start ovs_dpdk:

```
# ./datapath/dpdk/ovs-dpdk -c 0x0F -n 4 --proc-type primary --base-virtaddr=0x7ffb40000000 -- -p 0x03 --stats_core 0 --stats_int 5
```

7. Start the Open vSwitch daemon.

```
# ./ovs-vswitchd -c 0x100 -proc-type=secondary -- --pidfile=/tmp/vswitchd.pid
```

8. Using the ovs-ofctl utility, add flows entries to switch packets from port1 (Phy0) to port3 (Client 1) on the ingress path, and from port3 to port2 (Phy 1) on the egress path. The following example shows how to add flows for source and destination IPs 1.1.1.1 and 6.6.6.2.

```
# ./utilities/ovs-ofctl del-flows br0
# ./utilities/ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_src=1.1.1.1,nw_dst=6.6.6.2,
idle_timeout=0,action=output:3
# ./utilities/ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,nw_src=1.1.1.1,nw_dst=6.6.6.2,
idle_timeout=0,action=output:1
# ./utilities/ovs-ofctl dump-flows br0
```
9. Copy DPDK source and ovs_client to a shared directory to be passed to the VM:
   a. Create a share on host and copy DPDK and Intel® DPDK vSwitch client application to the share.
      
      ```
      # rm -rf /tmp/share
      # mkdir /tmp/share
      # mkdir /tmp/share/DPDK
      # chmod 777 /tmp/share
      # cp -aL $HOME/guest/ovs_client/* /tmp/share
      # cp -aL $HOME/DPDK-1.7/* /tmp/share/DPDK
      ```
   
10. Start the VM with IVSHMEM configuration:
   a. First, run `ovs-ivshm-mngr` with vm metadata (vm_1:port1).
      
      ```
      # /usr/src/dpdk-ovs/openvswitch/utilities/ovs-ivshm-mngr/build/ovs-ivshm-mngr
      -c 0x1 --proc-type=secondary -- vm_1:port1
      ```
   
   b. This command outputs something like this at the end:
      
      ```
      -device ivshmem, size=2048M, shm=fd:/dev/hugepages/rtemap_0:0x0:0x40000000:/dev/zero:0x0:0x3fffc000:/var/run/.dpdk_ivshmem_metadata_vm_1:0x0:0x4000
      ```
   
   c. Cut and paste this output to the `qemu` command to start the VM as follows:
      
      ```
      # cd <Intel DPDK vSwitch_INSTALL_DIR>
      # taskset 0x30 ./qemu/x86_64-softmmu/qemu-system-x86_64 -cpu host -boot c -smp 2 -hda /var/lib/libvirt/images/fedora20.qcow2 -m 4096M -vnc :2 -- enable-kvm -name 'client 1' -nographic -pidfile /tmp/vm1.pid -device ivshmem, size=2048M, shm=fd:/dev/hugepages/
      rtemap_0:0x0:0x40000000:/dev/zero:0x0:0x3fffc000:/var/run/.dpdk_ivshmem_metadata_vm_1:0x0:0x4000 -drive file=/tmp/share
      ```

B.3.6.2 VM Configuration for IVSHMEM Client Startup

1. Access the share drive and copy DPDK and ovs_client application.
   ```
   # mkdir -p /mnt/ovs_client
   # mkdir -p /root/ovs_client
   # mount -o iocharset=utf8 /dev/sdb1 /mnt/ovs_client
   # cp -a /mnt/ovs_client/* /root/ovs_client
   ```

2. Build DPDK on VM.
   ```
   # cd /root/ovs_client/DPDK
   # export RTE_SDK=/root/ovs_client/DPDK
   # export RTE_TARGET=x86_64-ivshmem-linuxapp-gcc
   # make uninstall
   # make install T=x86_64-ivshmem-linuxapp-gcc
   ```

B.3.6.2.1 IVSHMEM DPDK Client VM Application Startup

Build ovs_client and run it.
   ```
   # cd /root/ovs_client/
   # make clean
   # make
   # ./build/ovs_client -c 0x2 -n 4 -- -p port3
   ```
B.3.7 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.

The VM starts with two cores: vCPU0 and vCPU1. The Linux operating system and related tasks must use only vCPU0. vCPU1 is reserved to run the DPDK processes.

B.3.7.1 Affinitization Using Core Mask Parameter in the qemu, ovs-client and the test-pmd Startup Commands

The qemu, ovs_client and 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 both test-pmd and ovs-client commands. This ensures that the DPDK task application in the VM uses vCPU1. Ensure by running `top` command if vCPU1 is used at 100%.

However, with the qemu command, even though core mask is set to use two host cores for the VM’s vCPU0 and vCPU1, it allocates the vCPU0 and vCPU1 tasks on a single host core mostly the first core of the specified core mask. Hence, the qemu task needs to be re-affinitized.

B.3.7.2 Affinitized Host Cores for VMs vCPU0 and vCPU1

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

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

Ensure that the VM’s vCPU0 and vCPU1 are assigned to two separate host cores. For example:

- DPDK vSwitch uses cores 0, 1, 2 and 3 (`-c 0x0F`)
- 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>`)  

B.3.8 DPDK vSwitch Performance Tuning

Refer to the following pages for more information of DPDK vSwitch tuning:

- https://github.com/01org/dpdk-ovs/blob/96600eeec3df940d060f26713605f01f9db91c5/docs/07_Performance_Tuning.md#userspace-vhost-tuning
- https://github.com/01org/dpdk-ovs/blob/96600eeec3df940d060f26713605f01f9db91c5/docs/07_Performance_Tuning.md#ivshm-tuning
B.4 PCI Passthrough of a QAT Device

When setting up a QAT PCI device for a passthrough-to-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. Make sure the host has two QAT cards since we will set up an IPSec tunnel between two VMs, each VM using QAT to accelerate the tunnel. The VM uses standard Open vSwitch virtio+standard vhost IO virtualization method for networking.

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

   ```bash
   # 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.

   ```bash
   # echo 8086 0435 > /sys/bus/pci/drivers/pci-stub/new_id
   # 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
   
   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.

   ```bash
   # 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 pci-stub.

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

5. On a separate compute node that uses standard Open vSwitch for networking, add an ovs bridge called br0. The tap devices tap1, tap2, tap3 and tap4 is 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:

   ```bash
   # 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.

   ```bash
   # 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.

   ```bash
   # echo 0 > /proc/sys/net/ipv4/ip_forward
   ```
8. The two VMs can now be started with QAT devices as PCI passthrough. Following is a `qemu` command to pass 85:00.0 and 0c:00.0 to the VMs:

```
# 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 &
```

```
# 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 &
```

Refer to Appendix B.4.1 to setup a VM with QAT drivers, netkeyshim module and the strongSwan IPSec Software.

### B.4.1 VM Installation

#### B.4.1.1 Creating VM Image and VM Configuration

Refer to Appendix B.2.1.

#### B.4.1.2 Verifying Passthrough

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

```
# lspci -nn
```

Pass-through 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:

```
Ethernet controller [0200]: Intel Corporation Device [8086:0435]
```

#### B.4.1.3 Installing Intel® Communications Chipset 8900 to 8920 Series 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. Transfer the tarball using any preferred method. For example, USB memory stick, CDROM, or network transfer in the `/QAT` directory.

```
# tar -zxof <QAT_tarball_name>
```

There is a patch (`qat.patch`) that needs to be applied to be able to build the QAT drivers correctly on Linux kernel v3.15.6-200. You can download "Intel Quick Assist Technology patch for Netkey driver" zip file as documented in Section 4.1, "Obtaining Software Ingredients". The patch is zipped with this file and is called `qat.patch`. 

---

8. The two VMs can now be started with QAT devices as PCI passthrough. Following is a `qemu` command to pass 85:00.0 and 0c:00.0 to the VMs:

```
# 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 &
```

```
# 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 &
```

Refer to Appendix B.4.1 to setup a VM with QAT drivers, netkeyshim module and the strongSwan IPSec Software.

### B.4.1 VM Installation

#### B.4.1.1 Creating VM Image and VM Configuration

Refer to Appendix B.2.1.

#### B.4.1.2 Verifying Passthrough

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

```
# lspci -nn
```

Pass-through 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:

```
Ethernet controller [0200]: Intel Corporation Device [8086:0435]
```

#### B.4.1.3 Installing Intel® Communications Chipset 8900 to 8920 Series 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. Transfer the tarball using any preferred method. For example, USB memory stick, CDROM, or network transfer in the `/QAT` directory.

```
# tar -zxof <QAT_tarball_name>
```

There is a patch (`qat.patch`) that needs to be applied to be able to build the QAT drivers correctly on Linux kernel v3.15.6-200. You can download "Intel Quick Assist Technology patch for Netkey driver" zip file as documented in Section 4.1, "Obtaining Software Ingredients". The patch is zipped with this file and is called `qat.patch`. 

---

8. The two VMs can now be started with QAT devices as PCI passthrough. Following is a `qemu` command to pass 85:00.0 and 0c:00.0 to the VMs:

```
# 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 &
```

```
# 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 &
```

Refer to Appendix B.4.1 to setup a VM with QAT drivers, netkeyshim module and the strongSwan IPSec Software.

### B.4.1 VM Installation

#### B.4.1.1 Creating VM Image and VM Configuration

Refer to Appendix B.2.1.

#### B.4.1.2 Verifying Passthrough

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

```
# lspci -nn
```

Pass-through 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:

```
Ethernet controller [0200]: Intel Corporation Device [8086:0435]
```

#### B.4.1.3 Installing Intel® Communications Chipset 8900 to 8920 Series 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. Transfer the tarball using any preferred method. For example, USB memory stick, CDROM, or network transfer in the `/QAT` directory.

```
# tar -zxof <QAT_tarball_name>
```

There is a patch (`qat.patch`) that needs to be applied to be able to build the QAT drivers correctly on Linux kernel v3.15.6-200. You can download "Intel Quick Assist Technology patch for Netkey driver" zip file as documented in Section 4.1, "Obtaining Software Ingredients". The patch is zipped with this file and is called `qat.patch`. 

---
# cd /QAT/QAT1.6
# patch -b -p1 < <path_to_the_patch_location>

Verify that the patch is successfully applied.

2. Launch the script using the following command:

   
   # ./installer.sh

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

4. There is a known issue with the installation script in how it detects the QAT device on some Linux distributions, hence after executing step 4 run the following commands to complete the driver installation:

   
   # cd /QAT/QAT1.6/build
   # install -D -m 640 mmp_firmware.bin /lib/firmware/dh895xcc
   # install -D -m 640 mof_firmware.bin /lib/firmware/dh895xcc
   # install -D -m 640 icp_qa_al.ko /lib/modules/3.15.5/kernel/drivers/icp_qa_al.ko
   # /sbin/depmod -a
   # install -D -m 660 ../quickassist/config/dh895xcc_qa_dev0.conf.v2 /etc/dh895xcc_qa_dev0.conf
   # install -D -m 750 qat_service /etc/init.d/qat_service
   # install -D -m 750 adf_ctl /etc/init.d/adf_ctl
   # cp lib*_s.so /lib64
   # echo 'KERNEL=="icp_adf_ctl" MODE="0600"' > /etc/udev/rules.d/00-dh895xcc_qa.rules
   # echo 'KERNEL=="icp_dev[0-9]*" MODE="0600"' >> /etc/udev/rules.d/00-dh895xcc_qa.rules
   # echo 'KERNEL=="icp_dev_mem?" MODE="0600"' >> /etc/udev/rules.d/00-dh895xcc_qa.rules
   # chkconfig --add qat_service
   # /etc/init.d/qat_service start
   # /etc/init.d/qat_service status

5. Start/Stop acceleration software.

   
   # service qat_service start/stop

6. Setup the environment to install the Linux kernel crypto framework driver.

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

7. Unpack the Linux kernel crypto driver.

   
   # mkdir -p $ICP_ROOT/quickassist/shims/netkey
   # cd $ICP_ROOT/quickassist/shims/netkey
   # tar xzof <path_to>/icp_qat_netkey.L.<version>.tar.gz

8. Build the Linux kernel crypto driver.

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

9. Install the Linux kernel crypto driver.

   
   # cd $ICP_ROOT/quickassist/shims/netkey/icp_netkey
   # insmod ./icp_qat_netkey.ko
10. Verify that the module has been installed.

```
# lsmod | grep icp
```

The following is the expected output:

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

B.4.1.4 Setting Up IPSec Tunnel (on VMs)

The below diagram shows the IPSec tunnel setup configuration.

![Fig B-1 IPSec Tunnel Setup Configuration](image)

B.4.1.5 Installing strongSwan IPSec Software

Install strongSwan IPsec software on both VM1 and VM2.

1. Download the original strongswan-4.5.3 software package from the following link:
   
   `http://download.strongswan.org/strongswan-4.5.3.tar.gz`

2. Navigate to the `shims` directory:
   
   ```
   # cd $ICP_ROOT/quickassist/shims
   ```

3. Extract the source files from the strongSwan software package to the `shims` directory.
   
   ```
   # tar xzof <path_to>/strongswan-4.5.3.tar.gz
   ```

4. Navigate to the `strongSwan-4.5.3` directory.
   
   ```
   # cd $ICP_ROOT/quickassist/shims/strongswan-4.5.3
   ```

5. Configure strongSwan using the following command:
   
   ```
   # ./configure --prefix=/usr --sysconfdir=/etc
   ```

6. Build strongSwan using the following command:
   
   ```
   # make
   ```
7. Install strongSwan using the following command:

```shell
# make install
```
Repeat the installation of strongSwan IPsec software on the other VM.

### B.4.1.6 Configuring strongSwan IPsec Software

1. Add the following line to the configuration file `/etc/strongswan.conf` on the VM1 and VM2 platforms after the `charon {` line:

   ```
   load = curl aes des sha1 sha2 md5 pem pkcs1 gmp random x509 revocation hmac xcbc stroke kernel-netlink socket-raw updown
   ```

   After adding the line, the section looks like:

   ```
   # strongswan.conf - strongSwan configuration file
   charon {
   load = curl aes des sha1 sha2 md5 pem pkcs1 gmp random x509 revocation hmac xcbc stroke kernel-netlink socket-raw updown
   # number of worker threads in charon threads = 16
   ```

   **Note:** When the `/etc/strongswan.conf` text files are created, the line that starts with `load` and ends with `updown` must be the same line, despite the appearance of it being on two separate lines in this documentation.

2. Update the strongSwan configuration files on the VM1 platform:

   a. Edit `/etc/ipsec.conf`.

   ```
   # ipsec.conf - strongSwan IPsec configuration file
   # basic configuration
   config setup
   plutodebug=none
crlcheckinterval=180
strictcrlpolicy=no
nat_traversal=no
charonstart=no
plutostart=yes

conn %default
ikelifetime=60m
keylife=1m
rekeymargin=3m
keyingtries=1
keyexchange=ikev1
ike=aes128-sha-modp2048!
esp=aes128-sha1!

conn host-host
left=192.168.99.2
leftfirewall=no
right=192.168.99.3
auto=start
authby=secret
   ```

   b. Edit `/etc/ipsec.secrets` (this file might not exist and might need to be created).

   ```
   # /etc/ipsec.secrets - strongSwan IPsec secrets file
   192.168.99.2 192.168.99.3 : PSK "shared key"
   ```
3. Update the strongSwan configuration files on the VM2 platform:
   a. Edit `/etc/ipsec.conf`.
      
      ```
      # /etc/ipsec.conf - strongSwan IPsec configuration file

      config setup
      crlcheckinterval=180
      strictcrlpolicy=no
      plutostart=yes
      plutoctl=none
      charonstart=no
      nat_traversal=no

      conn %default
      ikelifetime=60m
      keylife=1m
      keyingtries=1
      keyexchange=ikev1
      ike=aes128-shal-modp2048!
      esp=aes128-shal!

      conn host-host
      left=192.168.99.3
      leftfirewall=no
      right=192.168.99.2
      auto=start authby=secret
      ```
   
   b. Edit `/etc/ipsec.secrets` (this file might not exist and might need to be created).
      
      ```
      # /etc/ipsec.secrets - strongSwan IPsec secrets file
      192.168.99.3 192.168.99.2 : PSK "shared key"
      ```

B.4.1.6.1 Starting strongSwan IPsec Software on the VM

IPSec tunnel is setup between two VMs as shown in figure in section Figure B-1 on page 80. The IPSec tunnel is setup for network interfaces on the subnet 192.168.99.0/24. The data networks for VM1 and VM2 are 1.1.1.0/24 and 6.6.6.0/24 respectively.

1. VM1 configuration settings:
   a. Network configuration:
      
      ```
      # cat /etc/sysconfig/network-scripts/ifcfg-eth1
      TYPE=Ethernet
      NAME=eth1
      BOOTPROTO=none
      ONBOOT=yes
      NETMASK=255.255.255.0
      IPADDR=1.1.1.2
      HWADDR=00:00:00:00:00:01

      # cat /etc/sysconfig/network-scripts/ifcfg-eth2
      TYPE=Ethernet
      NAME=eth2
      BOOTPROTO=none
      ONBOOT=yes
      NETMASK=255.255.255.0
      IPADDR=192.168.99.2
      HWADDR=00:00:00:00:00:02
      ```
   
   b. Add a route to the routing table for the VM2 network.
      
      ```
      # ip route add 6.6.6.0/24 dev eth2
      # route -n
      ```
c. Enable Linux Kernel forwarding and stop the firewall daemon.
   # echo 1 > /proc/sys/net/ipv4/ip_forward
   # service firewalld stop

d. Check QAT service status and insert the netkeyshim module.
   # export ICP_ROOT=/qat/QAT1.6
   # service qat_service status
   # insmod $ICP_ROOT/quickassist/shims/netkey/icp_netkey/icp_qat_netkey.ko

e. Start the IPSec process.
   # ipsec start

2. VM2 configuration settings:
   a. Network configuration:
      
      # cat /etc/sysconfig/network-scripts/ifcfg-eth1
      TYPE=Ethernet
      NAME=eth1
      BOOTPROTO=none
      ONBOOT=yes
      NETMASK=255.255.255.0
      IPADDR=6.6.6.6
      HWADDR=00:00:00:00:00:03
      
      # cat /etc/sysconfig/network-scripts/ifcfg-eth2
      TYPE=Ethernet
      NAME=eth2
      BOOTPROTO=none
      ONBOOT=yes
      NETMASK=255.255.255.0
      IPADDR=192.168.99.3
      HWADDR=00:00:00:00:00:04
      
   b. Add a route to the routing table for the VM2 network.
      # ip route add 1.1.1.0/24 dev eth2
   
   c. Enable Linux Kernel forwarding and stop the firewall daemon
      
      # echo 1 > /proc/sys/net/ipv4/ip_forward
      # service firewalld stop
   
   d. Check QAT service status and insert the netkeyshim module.
      
      # export ICP_ROOT=/qat/QAT1.6
      # service qat_service status
      # insmod $ICP_ROOT/quickassist/shims/netkey/icp_netkey/icp_qat_netkey.ko
   
   e. Start the IPSec process
      
      # ipsec start

3. Enable the IPsec tunnel “host-host” on both VM1 and VM2:
   
   # ipsec up host-host

4. To test the VM-VM IPsec tunnel performance, use Netperf.
Appendix C  Virtualized Broadband Network Gateway (BNG) Appliance Setup Instructions

This chapter presents step-by-step instructions that allow running BNG DPPD (DPDK Prototype Performance Demonstrator) application within the Sunrise Trail environment.

DPDK Prototype Performance Demonstrator application was analyzed and described in depth in the following documents that are publicly available:


The QoS functionality could also be tested within SRT environment using the same source code and the setup described below, though another configuration file should be used, and some environment modifications might be needed. The reference to the QoS sample is provided for informational purposes only, and QoS is not covered in this chapter.

For achieving the exemplary performance of the BNG application and minimizing the complexity of the task, the vSwitch was not included in the installation.

Due to dependencies, the operations listed in this chapter should be performed strictly in the sequence they are described below.

## C.1 Obtaining Software Ingredients

Software ingredients are generally available from relevant community sites.

<table>
<thead>
<tr>
<th>Software Component</th>
<th>Description</th>
<th>Patches</th>
<th>Link</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fedora 20</td>
<td>Standard Fedora 20 iso image</td>
<td></td>
<td><a href="http://download.fedoraproject.org/pub/fedora/linux/releases/20/Fedora/x86_64/iso/Fedora-">http://download.fedoraproject.org/pub/fedora/linux/releases/20/Fedora/x86_64/iso/Fedora-</a></td>
<td>Kernel v. 3.15.6-200.fc20</td>
</tr>
<tr>
<td>DPDK</td>
<td>DPDK poll mode driver, sample apps (bundled)</td>
<td>2 patches provided along with BNG DPPD package</td>
<td><a href="http://dpdk.org/browse/dpdk/snapshot/dpdk-1.7.0.tar.gz">http://dpdk.org/browse/dpdk/snapshot/dpdk-1.7.0.tar.gz</a></td>
<td>v.1.7.0</td>
</tr>
</tbody>
</table>
C.2 Hardware Requirements

Refer to the Sunrise Trail Hardware Requirements and BIOS settings as described in Section 3.0 and Section 5.1.1, respectively.

**Note:** Virtualized BNG Appliance requires for its functionality two Niantic 2x10 GbE cards installed in PCIe slots connected to the same CPU socket.

### C.2.1 BIOS Settings

<table>
<thead>
<tr>
<th>BIOS Section</th>
<th>Configuration</th>
<th>Setting for Compute Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Settings</td>
<td>Intel Turbo boost</td>
<td>Disabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>Enhanced Intel SpeedStep</td>
<td>Disabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>Processor C3</td>
<td>Disabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>Processor C6</td>
<td>Disabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>Intel Hyper-Threading Technology (HTT)</td>
<td>Enabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>Intel Virtualization Technology for Directed I/O (Vt-d)</td>
<td>Enabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>Pass-Through DMA support</td>
<td>Enabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>MLC Streamer</td>
<td>Enabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>MLC Spatial Prefetcher</td>
<td>Enabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>DCU Data Prefetcher</td>
<td>Enabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>DCU Instruction Prefetcher</td>
<td>Enabled</td>
</tr>
<tr>
<td>Processor Settings</td>
<td>Direct Cache Access (DCA)</td>
<td>Enabled</td>
</tr>
<tr>
<td>Power &amp; Performance</td>
<td>CPU Power and Performance Policy</td>
<td>Performance</td>
</tr>
<tr>
<td>Memory Configuration</td>
<td>Memory Power Optimization</td>
<td>Performance Optimized</td>
</tr>
<tr>
<td>Memory Configuration</td>
<td>Memory RAS and Performance Configuration -&gt; Numa Optimized</td>
<td>Enabled</td>
</tr>
<tr>
<td>System Acoustic &amp; Performance Configuration</td>
<td>Set Fan Performance</td>
<td>Performance</td>
</tr>
</tbody>
</table>

Table C-1  Software Ingredients (Continued)

<table>
<thead>
<tr>
<th>Software Component</th>
<th>Description</th>
<th>Patches</th>
<th>Link</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PktGen</td>
<td>Software Network Package Generator</td>
<td></td>
<td><a href="https://github.com/Pktgen/Pktgen-DPDK">https://github.com/Pktgen/Pktgen-DPDK</a></td>
<td>v2.7.1</td>
</tr>
<tr>
<td>BNG install scripts</td>
<td>Convenience scripts to install BNG</td>
<td>after-boot-host.sh run_VM.sh ts2.pl</td>
<td><a href="https://01.org/sites/default/files/page/vbng-scripts.zip">https://01.org/sites/default/files/page/vbng-scripts.zip</a></td>
<td></td>
</tr>
</tbody>
</table>
C.3 Installation

Download Fedora-20-x86_64-DVD.iso and create a bootable Fedora Core 20 USB thumb-drive following one of the options described here:

https://fedoraproject.org/wiki/How_to_create_and_use_Live_USB

Start FC20 installation from the USB drive.

C.3.1 Software Selection

Minimal install:
- Standard
- C Development Tools and Libraries
- Development tools
- RPM Development tools

Create the “stack” user account with administrative rights.

C.3.2 Host Post-Install

Note: Source IP addresses and paths in the examples below refer to the Intel CSIG Systems and Technology Lab in Belgium, and may differ in a particular environment. Other means of copying data may be used instead of scp.

1. From the “stack” account, execute the following command:
   `scp root@192.168.1.7:/data/Linux/fedora/20/Fedora/x86_64/iso/Fedora-20-x86_64-DVD.iso .`

2. As root, perform the following command:
   `yum update`

3. As root, disable SELinux.
   `setenforce 0`
   `vi /etc/selinux/config # -> SELINUX=disabled`

4. As root, disable the firewall.
   `systemctl disable firewalld.service`
   `reboot`
   `uname -a`
   `Linux srt10gz.mcplab.net 3.15.6-200.fc20.x86_64 #1 SMP Fri Jul 18 02:36:27 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux`
   `yum -y install vim`
   `yum -y install qemu-kvm`
   `yum -y install qemu-img`

5. As root, create a VM image:
   `qemu-img create -f qcow2 fc20min.qcow2 16G`
   Formatting 'fc20min.qcow2', fmt=qcow2 size=17179869184 encryption=off cluster_size=65536 lazy_refcounts=off
C.3.3 Network Configuration

Note: IP addresses are shown for example purposes only. Change the IP addresses as appropriate for the particular network environment.

1. Disable Fedora’s NetworkManager and enable “normal” networking.

   ```
   systemctl disable NetworkManager.service
   Computer\HKEY_LOCAL_MACHINE\SYSTEM\MountedDevices
   rm '/etc/systemd/system/dbus-org.freedesktop.nm-dispatcher.service'
   rm '/etc/systemd/system/dbus-org.freedesktop.NetworkManager.service'
   rm '/etc/systemd/system/multi-user.target.wants/NetworkManager.service'
   chkconfig network on
   systemctl restart network.service
   ```

2. Check `/etc/resolv.conf` for nameservers that were previously configured.

   ```
   cat /etc/resolv.conf
   Generated by NetworkManager
   nameserver 192.168.1.6
   nameserver 8.8.4.4
   ```

3. Check `/etc/sysconfig/network` for Default Gateway that was previously configured.

   ```
   cat /etc/sysconfig/network
   # Created by anaconda
   NETWORKING=yes
   HOSTNAME=SRT-FC20min
   GATEWAY=192.168.1.240
   ```

4. If Default Gateway is missing, change the IP address as appropriate for the network and run:

   ```
   echo "GATEWAY=192.168.1.240" >> /etc/sysconfig/network
   systemctl restart network.service
   ```

5. Check that `/etc/hostname` matches `/etc/sysconfig/network`.

   ```
   cat /etc/hostname
   SRT-FC20min
   ```

6. Configure the bridge.

   ```
   yum -y install tunctl
   (net-tools bridge-utils were already installed)
   ```

7. Configure the bridge. This bridge is used by the VM to get management network.

   ```
   vi /etc/sysconfig/network-scripts/ifcfg-em1
   DEVICE=em1
   ONBOOT=yes
   TYPE=Ethernet
   HWADDR="Put the real MAC address of network card here" # 00:1E:67:80:24:F5
   BRIDGE=br0
   ```

8. For a dynamic IP setup, use the following example:

   ```
   vi /etc/sysconfig/network-scripts/ifcfg-br0
   DEVICE=br0
   BOOTPROTO= dhcp
   ONBOOT=yes
   TYPE=Bridge
   ```
9. For a static IP setup, use the following example:

```
vi /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE="br0"
BOOTPROTO="static"
ONBOOT="yes"
TYPE="Bridge"
IPADDR=192.168.1.47
NETMASK=255.255.255.0
```

10. Restart the network.
```
    systemctl restart network.service
    systemctl -l status network.service
```

The following commands require net-tools, which already be installed (``sudo yum install net-tools``):
```
    tunctl -t tap1
    brctl addif br0 tap1
    ifconfig tap1 up
```

### C.3.4 Install the VM

Execute the following command:
```
/usr/bin/qemu-kvm \
-boot d -m 8192 -cpu host -enable-kvm -smp 10 -vnc :2 \n-hda fc20min.qcow2 \n-net nic,model=e1000,macaddr=00:1e:77:68:09:fd \n-net tap,ifname=tap1,script=no,downscript=no,vhost=on \n-cdrom /home/stack/Fedora-20-x86_64-DVD.iso
```

Choose the same initial setup settings as described for the host, using the whole HDD. Once installed, get back to the host.

### C.3.5 GRUB on the host

Execute the following command:
```
vi /etc/default/grub
```

```
... intel_iommu=on iommu=pt default_hugepagesz=1G hugepagesz=1G hugepages=8
isolcpus=10,11,12,13,14,15,16,17,18,19,30,31,32,33,34,35,36,37,38,39"
reboot
```

### C.3.6 Host Continued: PCI Passthrough and Starting the VM

1. Copy the following convenience scripts to the root home folder.
   - `after-boot-host.sh`
   - `run_VM.sh`
   - `ts2.pl`

2. Run the following script:
```
   ./after-boot-host.sh
```
3. Check that IOMMU functions properly.

dmesg | grep -i dmar
[ 0.000000] ACPI: DMAR 0000000cbdfa9618 000148 (v01 INTEL S2600GZ 06222004 INTL 20090903)
[ 0.109103] dmar: Host address width 46
[ 0.109561] dmar: DRHD base: 0x000000fbbfe000 flags: 0x0
[ 0.110031] dmar: IOMMU 0: reg_base_addr fbbfe000 ver 1:0 cap d2078c106f0466 ecap f020de
[ 0.110863] dmar: DRHD base: 0x000000ebff0c000 flags: 0x1
[ 0.111317] dmar: IOMMU 1: reg_base_addr ebff0c000 ver 1:0 cap d2078c106f0466 ecap f020de
[ 0.112150] dmar: RMRR base: 0x000000bdcf9000 end: 0x000000bdcf9000
[ 0.112612] dmar: ATSR flags: 0x0

dmesg | grep -i mmu
[ 0.000000] Command line: BOOT_IMAGE=/vmlinuz-3.14.5-200.fc20.x86_64 root=UUID=fd63109a-62ae-4e14-86f1-949f489f8e86 ro vconsole.font=latarcyrheb-sun16 rhgb ipv6.disable=1 intel_iommu=on iommu=pt
default_hugepagesz=1G hugepagesz=1G hugepages=8
[ 0.000000] Kernel command line: BOOT_IMAGE=/vmlinuz-3.14.5-200.fc20.x86_64 root=UUID=fd63109a-62ae-4e14-86f1-949f489f8e86 vconsole.font=latarcyrheb-sun16 rhgb ipv6.disable=1 intel_iommu=on iommu=pt
default_hugepagesz=1G hugepagesz=1G hugepages=8
[ 0.000000] Intel-IOMMU: enabled
[ 0.110031] dmar: IOMMU 0: reg_base_addr fbbfe000 ver 1:0 cap d2078c106f0466 ecap f020de
[ 0.111317] dmar: IOMMU 1: reg_base_addr ebff0c000 ver 1:0 cap d2078c106f0466 ecap f020de
[ 0.113179] IOAPIC id 2 under DRHD base 0xfbffe000 IOMMU 0
[ 0.113631] IOAPIC id 0 under DRHD base 0xebff0c000 IOMMU 1
[ 0.114083] IOAPIC id 1 under DRHD base 0xebff0c000 IOMMU 1
[ 1.429691] IOMMU 0 0xfbffe000: using Queued invalidation
[ 1.429450] IOMMU 1 0xebff0c000: using Queued invalidation
[ 1.430041] IOMMU: top
identity mapping for device 0000:00:00.0
[ 1.430511] IOMMU: hardware identity mapping for device 0000:00:01.0
...

4. Check where Niantic cards are.

lspci | grep Net
04:00.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
82:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
82:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
84:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
84:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)

Note: To start virtual machine and provide access to the NICs from within the VM using PCI passthrough, the following list of parameters needs to be passed to the qemu-kvm. Make sure the PCI addresses of Niantic cards in the run_VM.sh script are corresponding to the actual configuration, according to the list obtained by the command described above.

5. Start the VM.

./run_VM.sh

6. Affinitize QEMU threads to the second CPU socket.

./ts2.pl

C.3.7 Installation and Configuration Inside the VM

1. Execute the following command:

yum update

2. Disable SELinux.

setenforce 0
vi /etc/selinux/config # -> SELINUX=disabled

3. Disable the firewall.

systemctl disable firewalld.service
reboot
4. Execute the following command:

```bash
uname -a
```

```
Linux bras-vm.mcplab.net 3.15.6-200.fc20.x86_64 #1 SMP Fri Jul 18 02:36:27 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
```

C.3.8 GRUB in the VM

1. Execute the following command:

```bash
vi /etc/default/grub
```

```bash
... ipv6.disable=1 default_hugepagesz=1G hugepagesz=1G hugepages=2
isolcpus=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19 "
reboot
```

2. Verify that hugepages are available in the VM:

```bash
cat /proc/meminfo
```

```bash
... HugePages_Total:       2
HugePages_Free:        2
Hugepagesize:    1048576 kB
```

C.3.9 Install DPDK and BNG DPPD in the VM

1. Copy necessary helper scripts and configuration files to the VM:
   - after-boot-VM.sh
   - bind_to_igb_uio.sh

2. Download the DPDK and DPPD packages to the VM.

```bash
wget http://dpdk.org/browse/dpdk/snapshot/dpdk-1.7.0.tar.gz
```

3. Extract the dpdk archive.

```bash
tar xzvf dpdk-1.7.0.tar.gz
ln -s dpdk-1.7.0 DPDK
vi .bashrc
```

4. Add the following two lines to the end:

```bash
export RTE_SDK=$HOME/DPDK
export RTE_TARGET=x86_64-native-linuxapp-gcc
```

5. Re-login, or simply execute the following command:

```bash
. .bashrc
```

6. Extract DPPD BNG sources

```bash
unzip dppd-bng-v011.zip
```

7. Patch and build DPDK

```bash
cd DPDK
yum -y install kernel-devel
patch -p1 <dppd-BNG-v011/dpdk-patch/DPDK1.7-001-hash-bugfix.patch
patch -p1 <dppd-BNG-v011/dpdk-patch/DPDK1.7-002-hash-features.patch
make install T=x86_64-native-linuxapp-gcc
```
modprobe uio

insmod $RTE_SDK/$RTE_TARGET/kmod/igb_uio.ko
cd

Check the PCI addresses of the Niantic cards:

```
lspci | grep Net
```

00:04.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
00:05.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
00:06.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
00:07.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)

8. Ensure that the correct PCI addresses are listed in the script `bind_to_igb_uio.sh`, then run the following script:

```
./after-boot-VM.sh
```

9. Build BNG DPPD application.

```
yum -y install ncurses-devel
cd dppd-BNG-v011
make
```

10. Ensure that the application starts.

```
./build/dppd -f config/bras.cfg
```

11. Exit the application by pressing **ESC** or **CTRL-C**.

## C.4 PktGen

WinDriver Network Traffic Generator functionality was extended to give PktGen the capability to generate the test-load patterns necessary for testing the BNG application.

The Pktgen project is maintained on Github at:

```
https://github.com/Pktgen/Pktgen-DPDK
```

Following is the direct download link to the sources:

```
https://github.com/Pktgen/Pktgen-DPDK/archive/master.zip
```

## C.4.1 Pktgen-DPDK Host Requirements

Pktgen must run on a separate host. The hardware and OS setup are identical to the Sunrise Trail platform, and can be used.

All the installation steps described above for vBNG VM are equally applicable for the PktGen host machine installation process.

## C.4.2 Get the Pktgen-DPDK Source

Get the source from Github:

```
git clone https://github.com/Pktgen/Pktgen-DPDK.git
cd Pktgen-DPDK
```
C.4.3 Build Bundled DPDK and Pktgen

1. In addition to the Sunrise Trail system installation described throughout Appendix C.3, an extra package must be installed for Pktgen to compile correctly:

   ```bash
   yum -y install libpcap-devel
   ```

2. Pktgen comes with its own distribution of the DPDK sources. This bundled version of DPDK must be used: it contains some WindRiver specific helper libraries that are not in the default DPDK distribution, which Pktgen depends on.

   The `$RTE_TARGET` variable must be set to a specific value, otherwise these libraries will not build.

   ```bash
   cd
   ln -fs /root/Pktgen-DPDK/dpdk DPDK
   vi .bashrc
   ```

3. Add the following two lines to the end:

   ```bash
   export RTE_SDK=$HOME/DPDK
   export RTE_TARGET=x86_64-pktgen-linuxapp-gcc
   ```

4. Re-login, or simply execute the following command:

   ```bash
   . .bashrc
   ```

5. Build the basic DPDK libraries and extra helpers.

   ```bash
   cd $RTE_SDK
   make install T=$RTE_TARGET
   ```


   ```bash
   cd examples/pktgen
   make
   ```

7. Copy the following helper scripts:

   - `after-boot-VM.sh`
   - `bind_to_igb_uio.sh`

8. Adapt `bind_to_igb_uio.sh` script accordingly to the actual NICs in use.

   ```bash
   ./after-boot-VM.sh
   ```

C.4.4 Run Pktgen

1. Copy necessary helper scripts and configuration files to the traffic generator machine in the `$RTE_SDK/examples/pktgen` folder:

   - `run_pktgen_BNG_load.sh`
   - `BNG-config.pkt`

2. Run Pktgen with the following command line (change blacklist, coremask, portmask, matrix parameters as needed):

   ```bash
   cd $RTE_SDK/examples/pktgen
   ./run_pktgen_BNG_load.sh
   ```
C.4.5 Troubleshooting

<table>
<thead>
<tr>
<th>Observed Issue</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pktgen application is not found when run_pktgen_BNG_load.sh script is executed.</td>
<td>Change ./app/build/app/pktgen in the run_pktgen_BNG_load.sh script to ./app/pktgen.</td>
</tr>
<tr>
<td>Pktgen application runs, but the screen does not display anything (remains black).</td>
<td>Add (or remove) &quot;-T&quot; parameter right after double dash in the pktgen command line. For example: sudo ./app/pktgen -c 1ff -n 4 -- -T -p 0x285 -m &quot;[1:2].0, [3:4].1, [5:6].2, [7:8].3&quot; -f BNG-config.pkt</td>
</tr>
<tr>
<td>BNG application gets cluttered screen or complains about not recognized packets.</td>
<td>Swap 10 GbE interconnect cables so the connection diagram corresponds to the one described in the BNG-config.pkt file.</td>
</tr>
<tr>
<td>Performance is below 10.6 Mpps. One channel receives 1/10 of expected traffic.</td>
<td>In the pktgen command line type the following command: set 0,2 rate 100</td>
</tr>
</tbody>
</table>

C.5 Performance Results

The performance of the vBNG Appliance has been verified in a worst-case scenario using the smallest packets size. The obtained performance numbers correspond closely to the data published in the Intel Network Builders Reference Architecture: Network Function Virtualization: Virtualized BRAS with Linux* and Intel® Architecture, available at:

http://networkbuilders.intel.com/solutionslibrary

Following is the direct download link to the Reference Architecture:

http://networkbuilders.intel.com/docs/Network_Builders_RA_vBRAS_Final.pdf

In the trial the Sunrise Trail virtualized BNG appliance attained 10.9 Mpps transmitted on each of all four ports, resulting of 43.6 Mpps of a total transmit packet rate (conservative measurements).
Appendix D  Additional OpenDaylight Information

D.1  Alternate Method for Disabling the OVSDB Neutron Plug-in

Instead of disabling the OVSDB neutron service by removing the OVSDB neutron bundle file, it is possible to disable the bundle from the OSGi console. However, there does not appear to be a way to make this persistent, so it must be done each time the controller restarts.

Once the controller is up and running, connect to the OSGi console. The `ss` command displays all of the bundles that are installed and their status. Adding a string(s) filters the list of bundles. List the OVSDB bundles:

```
osgi> ss ovs
"Framework is launched."

id    State       Bundle
106    ACTIVE      org.opendaylight.ovsdb.northbound_0.5.0
112    ACTIVE      org.opendaylight.ovsdb_0.5.0
262    ACTIVE      org.opendaylight.ovsdb.neutron_0.5.0
```

Note that there are three OVSDB bundles running (the OVSDB neutron bundle has not been removed in this case).

Disable the OVSDB neutron bundle and then list the OVSDB bundles again:

```
osgi> stop 262
osgi> ss ovs
"Framework is launched."

id    State       Bundle
106    ACTIVE      org.opendaylight.ovsdb.northbound_0.5.0
112    ACTIVE      org.opendaylight.ovsdb_0.5.0
262    RESOLVED    org.opendaylight.ovsdb.neutron_0.5.0
```

Now the OVSDB neutron bundle is in the RESOLVED state, which means that it is not active.
D.2 Creating Flows with OpenDaylight

The OpenDaylight web GUI can be used to create OpenFlow flows in the OVS host bridge instances. The steps to create a simple rule that forwards all packets received on one port to another port are as follows:

1. Select the **Flows** tab on the main screen.
2. Press **Add Flow Entry**.
3. Fill in the fields in the **Add Flow Entry** pop-up window:
   a. Enter a name for the flow.
   b. Select **Node** from the drop-down box.
   c. Select **Input Port** from the drop-down box.
   d. Modify **Priority** if desired.
   e. Scrolling down, clear the **Ethernet Type** field (for this rule).
   f. Scroll further down and select **Add Output Port** entry from the **Actions** drop-down box.

![Figure D-1 Flow Entry Pop-up (Selecting an Input Port)]
g. Select the desired output port.

h. Press **Install Flow** to finish.

The following diagram shows the main flow screen after several flows are created.

![Flow Screen](image)

**Figure D-3  Flow Screen**
Dumping the flows directly on the OVS host shows the following:

```
# ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=222.015s, table=0, n_packets=0, n_bytes=0, idle_age=222,
priority=500,in_port=16 actions=output:80
  cookie=0x0, duration=159.067s, table=0, n_packets=0, n_bytes=0, idle_age=159,
priority=500,in_port=81 actions=output:82
  cookie=0x0, duration=28.489s, table=0, n_packets=0, n_bytes=0, idle_age=28,
priority=500,in_port=80 actions=output:16
  cookie=0x0, duration=70.754s, table=0, n_packets=0, n_bytes=0, idle_age=70,
priority=500,in_port=82 actions=output:81
```

More complex rules can be created by setting Ethernet Type and source and destination MAC and IP addresses. Note that the DPDK vSwitch has some limitations on the kinds of flow entries that can be created (refer to the DPDK vSwitch documentation).

Flows can be managed by clicking on a flow in the Flow Entries section (upper-left of screen) and then pressing one of the buttons in the Flow Detail section (bottom-right of screen). Flows can be edited, removed, or uninstalled. Uninstalling a flow keeps the flow in the OpenDaylight data store but removes it from the flow tables in the OVS host. An uninstalled flow can be reinstalled (possibly after editing it).

## D.3 Other Tips for Running OpenDaylight

### D.3.1 Running the Controller Directly

Another way to get the OSGi console is to run the controller directly instead of using `systemctl` to start it. In this case, the controller should first be stopped via `systemctl` as previously described.

In a dedicated terminal window, do the following steps:

```
# cd /usr/share/opendaylight-controller
# export ODL_BASEDIR=/var/lib/opendaylight-controller
# export ODL_DATADIR=/var/lib/opendaylight-controller
# ./run.odl.sh virt-ovsdb
```

This brings up the controller and the OSGi console in the terminal window. All of the log messages and Java exceptions are also displayed on this screen as they occur, which is the reason for using this method of running the controller. Type `Exit` to leave the OSGi console and terminate the controller.

### D.3.2 Controller No Longer Starts via systemctl After Running it Directly

Starting and stopping the controller directly (as previously described) appears to leave the system in a state where it can no longer be started via the systemctl method.

The program `journalctl` (for example, `# journalctl -r`) assisted in showing the log of errors that were occurring as the OpenDaylight controller was failing to start. In this case, the log indicated that the `/var/lib/opendaylight-controller/configuration/org.eclipse.osgi` directory was not getting cleaned up. The problem was resolved by changing to that directory and removing all the contents (including hidden files and directories).
Following is what the contents of what the directory looked like:

```
# cd /var/lib/opendaylight-controller/configuration/org.eclipse.osgi
# ls -ltr
total 1232
  drwxr-xr-x  4 opendaylight opendaylight    4096 Apr 16 10:56 ..
  drwxr-xr-x 253 opendaylight opendaylight    4096 Apr 16 10:56 bundles
-rw-r--r--   1 opendaylight opendaylight   99591 Apr 16 10:56 .bundledata.1
-rw-r--r--   1 opendaylight opendaylight 1100098 Apr 16 10:56 .lazy.1
-rw-r--r--   1 opendaylight opendaylight 39907 Apr 16 10:56 .state.1
  drwxr-xr-x   4 opendaylight opendaylight    4096 Apr 16 10:56 .
  drwxr-xr-x   2 opendaylight opendaylight    4096 Apr 16 10:56 .manager
```

Removing the three hidden files and the bundles and .manager directory restored the ability to start the controller with `systemctl`. 
### Appendix E  Glossary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR</td>
<td>Application Targeted Routing</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial OffThe-Shelf</td>
</tr>
<tr>
<td>DPI</td>
<td>Deep Packet Inspection</td>
</tr>
<tr>
<td>FCS</td>
<td>Frame Check Sequence</td>
</tr>
<tr>
<td>GRE</td>
<td>Generic Routing Encapsulation</td>
</tr>
<tr>
<td>GRO</td>
<td>Generic Receive Offload</td>
</tr>
<tr>
<td>IOMMU</td>
<td>Input/Output Memory Management Unit</td>
</tr>
<tr>
<td>Kpps</td>
<td>Kilo packets per seconds</td>
</tr>
<tr>
<td>KVM</td>
<td>Kernel-based Virtual Machine</td>
</tr>
<tr>
<td>LRO</td>
<td>Large Receive Offload</td>
</tr>
<tr>
<td>MSI</td>
<td>Message Signaling Interrupt</td>
</tr>
<tr>
<td>MPLS</td>
<td>Multi-protocol Label Switching</td>
</tr>
<tr>
<td>Mpps</td>
<td>Millions packets per seconds</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Interface Card</td>
</tr>
<tr>
<td>pps</td>
<td>Packets per seconds</td>
</tr>
<tr>
<td>QAT</td>
<td>Quick Assist Technology</td>
</tr>
<tr>
<td>QinQ</td>
<td>VLAN stacking (802.1ad)</td>
</tr>
<tr>
<td>RA</td>
<td>Reference Architecture</td>
</tr>
<tr>
<td>RSC</td>
<td>Receive Side Coalescing</td>
</tr>
<tr>
<td>RSS</td>
<td>Receive Side Scaling</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider</td>
</tr>
<tr>
<td>SR-IOV</td>
<td>Single root I/O Virtualization</td>
</tr>
<tr>
<td>TCO</td>
<td>Total Cost of Ownership</td>
</tr>
<tr>
<td>TSO</td>
<td>TCP Segmentation Offload</td>
</tr>
</tbody>
</table>
## Appendix F  References

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux man pages</td>
<td>Man taskset</td>
</tr>
<tr>
<td>Linux kernel Documentation</td>
<td>Documentation/IRQ-affinity.txt</td>
</tr>
<tr>
<td>Intel® DPDK vSwitch</td>
<td><a href="https://01.org/packet-processing">https://01.org/packet-processing</a></td>
</tr>
</tbody>
</table>
NOTE: This page intentionally left blank.