Intel® Data Plane Development Kit (Intel® DPDK) vSwitch

Getting Started Guide

December 2013
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 2013</td>
<td>0.1.0</td>
<td>Initial version</td>
</tr>
<tr>
<td>April 2013</td>
<td>0.2.0</td>
<td>Updated with sections for IVSHM and 12-Tuple support</td>
</tr>
<tr>
<td>May 2013</td>
<td>0.3.0</td>
<td>Removed vswitchd instructions and changed ovs_dpdk parameters</td>
</tr>
<tr>
<td>June 2013</td>
<td>0.4.0</td>
<td>Updated to reflect directory naming changes and merge of QEMU</td>
</tr>
<tr>
<td>June 2013</td>
<td>0.5.0</td>
<td>Updated to add details of KNI configuration</td>
</tr>
<tr>
<td>June 2013</td>
<td>0.6.0</td>
<td>Updated to add Wind River* Linux* details and update method of mapping hugepages in the guest for IVSHM</td>
</tr>
<tr>
<td>June 2013</td>
<td>0.7.0</td>
<td>Document restructuring and rework</td>
</tr>
<tr>
<td>July 2013</td>
<td>0.8.0</td>
<td>Updated to add changes for flow manipulation code</td>
</tr>
<tr>
<td>July 2013</td>
<td>0.9.0</td>
<td>Updates to flow manipulation sections and correction of errata</td>
</tr>
<tr>
<td>August 2013</td>
<td>0.9.1</td>
<td>Fixed broken links and added additional sample test setups</td>
</tr>
<tr>
<td>October 2013</td>
<td>1.0</td>
<td>Minor corrections and conversion to FrameMaker</td>
</tr>
<tr>
<td>October 2013</td>
<td>1.0.1</td>
<td>Minor updates related to 802.1Q VLAN packet support</td>
</tr>
<tr>
<td>November 2013</td>
<td>001</td>
<td>New public document number assigned with new revision number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minor updates related to multiple actions and strip VLAN, mod VLAN, and drop action support</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Significant rewrite of Wind River* Linux* section</td>
</tr>
<tr>
<td>December 2013</td>
<td>002</td>
<td>Minor updates related to ovs-testsuite, OFTest and Intel® DPDK v1.5 support</td>
</tr>
</tbody>
</table>
# Contents

## 1.0 Introduction
  1.1 Description of Release ................................................................. 7
  1.2 Terminology .............................................................................. 7

## 2.0 Release Notes and Known Bugs
  2.1 Supported Operating Systems ...................................................... 8
  2.2 Supported Processors .................................................................. 8
  2.3 Intel® DPDK vSwitch ................................................................. 8
  2.4 Intel® DPDK vSwitch Sample Guest Application ......................... 10
  2.5 Open vSwitch ........................................................................... 10
  2.6 Open vSwitch Commands .......................................................... 10
  2.7 QEMU ....................................................................................... 10
  2.8 vswitchd ................................................................................... 10
  2.9 ovs-vsect .................................................................................. 11
  2.10 ovs-ofctl ............................................................................... 11
  2.11 ovs-dpctl ............................................................................... 11

## 3.0 System Requirements ................................................................. 12
  3.1 Required Libraries ...................................................................... 12
  3.2 Downloading Intel® DPDK .......................................................... 12

## 4.0 Compiling Intel® DPDK vSwitch from Source .............................. 14
  4.1 Compile the Intel® DPDK .......................................................... 14
  4.2 Compile Open vSwitch ............................................................. 14
  4.3 Compile QEMU ......................................................................... 14

## 5.0 Compiling and Running Sample Applications .............................. 15
  5.1 VirtIO ....................................................................................... 15
     5.1.1 Host Setup ......................................................................... 15
        5.1.1.1 Configure Kernel Boot Parameters ................................ 15
        5.1.1.2 Build Source Code ..................................................... 15
        5.1.1.3 Setup Intel® DPDK ..................................................... 15
        5.1.1.4 Add Ports to the vSwitch ........................................... 15
        5.1.1.5 Start Intel® DPDK vSwitch (ovs_dpdk) ....................... 15
        5.1.1.6 Start QEMU .............................................................. 16
        5.1.1.7 Program the Switch’s Flow Tables ............................. 16
     5.1.2 Guest Setup ....................................................................... 16
        5.1.2.1 Ensure VirtIO Ethernet Device Is Present ................. 17
        5.1.2.2 Configure VM Network Interfaces ............................ 17
     5.2 IVSHM Setup ......................................................................... 17
        5.2.1 Host Setup ..................................................................... 17
           5.2.1.1 Configure Kernel Boot Parameters ......................... 17
           5.2.1.2 Build Source Code ................................................. 17
           5.2.1.3 Set Up the Intel® DPDK ........................................ 18
           5.2.1.4 Add Ports to the vSwitch ..................................... 18
           5.2.1.5 Start Intel® DPDK vSwitch (ovs_dpdk) ............... 18
           5.2.1.6 Program the Switch’s Flow Tables ..................... 19
           5.2.1.7 Copy Required Files to a Temporary Location ....... 19
           5.2.1.8 Start QEMU ......................................................... 19
        5.2.2 Guest Configuration ........................................................ 19
           5.2.2.1 Enable Hugepages ............................................... 20
           5.2.2.2 Obtain PCI Device ID of Mapped Memory ............ 20
           5.2.2.3 Link Intel® DPDK Hugepage in the Guest ............ 20
           5.2.2.4 Copy Required Files from Host ............................ 20
           5.2.2.5 Compile Intel® DPDK ............................................ 20
Tables
1  Terms...................................................................................................................... 7
2  Affinitizing the Host Core ....................................................................................23
3  Flow Strings Supported with ovs-ofctl............................................................... 35
4  Arguments Supported with ovs-dpctl................................................................. 38
5  Arguments Required with ovs-dpctl....................................................................... 40
6  Valid Port Value..................................................................................................... 50
1.0 Introduction

This document contains detailed instructions for building and running the Intel® Data Plane Development Kit (Intel® DPDK) vSwitch software. It describes how to compile and run Intel® DPDK vSwitch, QEMU, and guest applications in a Linux* environment.

1.1 Description of Release

This guide covers modified Open vSwitch and QEMU packages that enable the use of Intel® DPDK to demonstrate performance and to be used as a reference architecture. This release ports Intel® DPDK vSwitch to Intel® DPDK 1.5.2, adds partial support for the integrated Open vSwitch testsuite and OFTest, and adds support for virtual Ethernet devices. Additionally, it incorporates a number of bug fixes and significant performance improvements. Refer to the NEWS file included in the release package for additional details.

Caution: Please note that the software in this release is under various open source licenses and, as such, is provided “as is” and without warranty. Intel is not liable for any damages arising from the use of this software.

1.2 Terminology

Definitions of terms used in this document are listed in Table 1.

Table 1. Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARP</td>
<td>Address Resolution Protocol</td>
</tr>
<tr>
<td>DPDK</td>
<td>Data Plane Development Kit</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In, First Out</td>
</tr>
<tr>
<td>IVSHM</td>
<td>Inter VM Shared Memory</td>
</tr>
<tr>
<td>KNI</td>
<td>Kernel Network Interface</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MTU</td>
<td>Maximum Transmission Unit</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Interface Card</td>
</tr>
<tr>
<td>OVS</td>
<td>Open vSwitch</td>
</tr>
<tr>
<td>QEMU</td>
<td>Quick Emulator</td>
</tr>
<tr>
<td>PCI</td>
<td>Peripheral Connect Interface</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
</tbody>
</table>

§ §
2.0 Release Notes and Known Bugs

2.1 Supported Operating Systems

This release has been validated against the following operating systems.

- Host OS
  - Wind River* Linux* 5.0.1.0_standard – Kernel 3.4.34
  - Fedora* 16 – Kernel 3.6.7-4
- Guest OS
  - Fedora* 16 – Kernel 3.6.7-4

2.2 Supported Processors

This release has been validated on the Intel® Xeon® Processor E5 Family.

2.3 Intel® DPDK vSwitch

- This release supports Intel® DPDK v1.5.2 only. Intel® DPDK v1.4.0 is no longer supported.
- QEMU is added as an Intel® DPDK secondary process – attempting to run a secondary process before a primary process will result in a segfault. This is standard Intel® DPDK behavior.
- Intel® Virtualization Technology for Directed I/O (Intel® VT-d) should be disabled in the BIOS settings, unless PCI passthrough is required, in which case the following options should be added to the kernel boot parameters:
  
  ```
  intel_iommu=on iommu=pt
  ```

- Memory corruption is possible if the cores specified using the `-c` option overlap between processes.
- When starting the VMs, the following warning may appear:

  (ASLR) is enabled in the kernel. This may cause issues with mapping memory into secondary processes. Although in most cases this warning is harmless, to suppress it, run the following command:

  ```
  # echo 0 > /proc/sys/kernel/randomize_va_space
  ```

- Only one instance of the `kni_client` application should be started in a guest; however, to create multiple KNI devices in a single VM, use the `-p` parameter, a bitmask that specifies the KNI queues to initialize and connect to, to specify the KNI devices to be created. For example, to initialize KNI queues 0 and 1 in the VM, the `-p` value is 3 (see KNI section for further details):

  ```
  ./kni_client -c 0x1 --proc-type=secondary -- -p 0x3
  ```

- In Intel® DPDK vSwitch, packet data is copied before it is injected into VirtIO, which may introduce a higher packet drop rate with larger packet sizes. In general,
throughput speeds for VirtIO are similar to standard QEMU, if slightly lower; currently, ways to improve the performance with a different design are being investigated. KNI is offered as a backwards-compatible alternative to VirtIO (i.e., it supports non-Intel® DPDK userspace applications in the guest), and offers significantly better performance compared to VirtIO. Intel recommends this option when high throughput is required in a non-Intel® DPDK application use case.

- This release has not been tested or validated for use with Virtual Functions, although it should theoretically work with Intel® DPDK v1.5.2.
- If testing performance with TCP, variances in performance may be observed; this variation is due to the protocol's congestion-control mechanisms. UDP produces more reliable and repeatable results, and it is the preferred protocol for performance testing.
- On start-up, Intel® DPDK vSwitch may issue an error:

```
EAL: memzone_reserve_aligned_thread_unsafe(): memzone
<RG_MP_log_history> already exists
RING: Cannot reserve memory
```

When an Intel® DPDK process starts, it attempts to reserve memory for various rings through a call to `rte_memzone_reserve`; in the case of an Intel® DPDK primary process, the operation should succeed, but for a secondary process, it is expected to fail, as the memory has already been reserved by the primary process. The particular ring specified in the error message - `RG_MP_log_history` - does not affect operation of the secondary process, so this error may be disregarded.

- On start-up, `ovs_dpdk` may complain that no ports are available (when using an Intel® DPDK-supported NIC):

```
Total ports: 0
EAL: Error - exiting with code: 1
Cause: Cannot allocate memory for port tx_q details
```

These error messages indicate that Intel® DPDK initialization failed because it did not detect any recognized physical ports. One possible cause is that the NIC is still driven by the default `ixgbe` driver. To resolve this issue, run `DPDK/tools/pci_unbind.py` before starting `ovs-dpdk`. (This process lets the Intel® DPDK poll mode driver take over the NIC.) For example, `pci_unbind.py -b igb_uio <PCI ID of NIC port>` binds the NIC to the Intel® DPDK `igb_uio` driver.

- As `ovs_dpdk` requires modification to achieve compatibility with 82571EB-based dual-port cards, modify `openvswitch/datapath/dpdk/init.c`, updating the value of `tx_rings` in the `init_port` function from `num_clients` to 1, and recompile.
- Passing a VLAN packet with VLAN ID 0, but a priority greater than 0 (A priority tagged packet) is not currently supported, and passing this type of packet will render the switch unresponsive.
- Introducing traffic into the switch, prior to starting the vswitch daemon may cause undesired behaviour. This issue is being investigated and will be resolved in a future release.
2.4 Intel® DPDK vSwitch Sample Guest Application

- In the current IVSHM implementation, a single Intel® DPDK hugepage is shared between all guests, with the implication that it may be possible for VMs to access each other's memory. Consequently, IVSHM is intended for use only when applications in VMs are trusted.

2.5 Open vSwitch

Open vSwitch builds with a number of warnings (of type deprecated declaration), originating from the original Open Source Open vSwitch v1.5.0 release package.

2.6 Open vSwitch Commands

Only a subset of the OpenFlow actions implemented in the original Open Source Open vSwitch application are currently supported by Intel® DPDK vSwitch. Currently, these are the following:

- output
- drop
- VLAN actions
  - strip_vlan
  - mod_vlan_vid
  - mod_vlan_pcp

As of release 0.7, Intel® DPDK vSwitch adds support for action sets.

*Note:* strip_vlan should be used in conjunction with the output action. If this is not done, the packet will be dropped after applying the action. This is standard OVS behavior.

2.7 Open vSwitch Testsuite

Open vSwitch contains a number of unit tests which collectively form the OVS "testsuite". While the majority of these tests currently pass without issue, a small number do fail. The common cause of failure for these tests is a discrepancy in the command line arguments required by many of the utilities in standard Open vSwitch and their equivalents in Intel® DPDK vSwitch. In addition, test three (3) causes the testsuite to hang and should be skipped. These issues will be resolved in a future release.

Many of the tests also fail due to differences in the required parameters for utilities such as ovs-dpctl (i.e. Intel® DPDK vSwitch’s version of these utilities require EAL parameters).

In addition to the standard unit tests, Intel® DPDK vSwitch extends the testsuite with a number of ‘Intel® DPDK vSwitch’-specific unit tests. These tests require root privileges to run, due to the use of hugepages by the Intel® DPDK library.

2.8 OFTest

Adding a route when using virtual Ethernet devices has been known to cause system instability. The root cause of this issue is currently being investigated.
A number of OFTest tests currently fail. In most cases this is due to missing functionality in either standard Open vSwitch or Intel® DPDK vSwitch. These will be resolved as additional functionality is added.

OFTest support has been validated against Scapy v2.2

2.9 QEMU

IVSHM model has been validated only on QEMU v1.4.0 and above. This is due to a known bug in earlier versions (such as v1.1), which prevents mapping of hugepages of size > 256 MB (1-GB hugepage is used in IVSHM).

2.10 ovs-vswitchd

Not all functionality that is supported by Open vSwitch is supported by the Intel® DPDK vSwitch.

2.11 ovs-vsctl

Not all functionality that is supported by Open vSwitch is supported by the Intel® DPDK vSwitch.

2.12 ovs-ofctl

- Not all functionality that is supported by Open vSwitch is supported by the Intel® DPDK vSwitch.
- The only OpenFlow actions currently supported are those detailed in Section 2.6

2.13 ovs-dpctl

- Not all functionality that is supported by Open vSwitch is supported by the Intel® DPDK vSwitch.
- The only OpenFlow actions currently supported are those detailed in Section 2.6

§ §
3.0 System Requirements

This section describes how to build and run Intel® DPDK vSwitch, Open vSwitch, QEMU, and sample Intel® DPDK guest applications.

3.1 Required Libraries

The following libraries are needed to compile the various components within the release:

- gcc
- kernel-dev
- kernel-devel
- coreutils
- make
- nasm
- glibc-devel.i686
- libc6-dev-i386
- glibc-devel.x64_86
- glibc-devel
- kernel-devel-3.3.4-5.fc17.x86_64 (matching kernel)
- autoconf
- automate
- autom4te
- automake
- zlib-devel
- glib2-devel.x86_64
- libtool

3.2 Download Intel® DPDK

To download the Intel® DPDK release package DPDK.L.1.5.2-2.zip:

- Existing customers can download the release package from the Intel Business Portal website: http://www.intel.com/ibl.
  - Under Information Desk/Design Kits, select the Embedded category, under Platforms and Solutions.
  - Under Development Kits, select Intel Data Plane Development Kit (Intel DPDK), then select Embedded Software: Intel Data Plane Development Kit - Technical.
— Download Intel® Data Plane Development Kit (Intel® DPDK) - Release 1.5.2 (L1.5.2-2) - Code (Zip File) and Intel® Data Plane Development Kit (Intel® DPDK) - Release 1.5.2 - Documentation.

• Otherwise, please register on the Intel® Embedded website, and request design assistance for your project.

§ §
4.0 Compiling Intel® DPDK vSwitch from Source

4.1 Compile the Intel® DPDK

Expand the Intel® DPDK release package:

```bash
mkdir ovs_dpdk
 tar -C ovs_dpdk -xzvf <release_pkg_name>.tar.gz
```

Modify the Intel® DPDK buildsystem so that libraries are position independent:

```bash
--- a/mk/target/generic/rte.vars.mk
+++ b/mk/target/generic/rte.vars.mk
@@ -105,7 +105,7 @@ ifeq ($(KERNELRELEASE),
 # merge all CFLAGS
 CFLAGS := $(CPU_CFLAGS) $(EXECENV_CFLAGS) $(TOOLCHAIN_CFLAGS) $(MACHINE_CFLAGS)
- CFLAGS += $(TARGET_CFLAGS)
+ CFLAGS += $(TARGET_CFLAGS) -fPIC

 # merge all LDFLAGS
 CFLAGS := $(CPU_CFLAGS) $(EXECENV_CFLAGS) $(TOOLCHAIN_CFLAGS) $(MACHINE_CFLAGS)
- LDFLAGS += $(TARGET_LDFLAGS)
+ LDFLAGS += $(TARGET_LDFLAGS) -fPIC
```

Compile the Intel® DPDK. Please refer to the Intel® Data Plane Development Kit (Intel® DPDK) - Getting Started Guide (Document Number: 326002) for instructions on performing this step.

4.2 Compile Open vSwitch

Once the Intel® DPDK is built, perform the following steps to build Open vSwitch:

```bash
cd openvswitch
./boot.sh
./configure RTE_SDK=/path/to/dpdk
make
```

At this point it is recommended that unit tests are run to ensure correct installation - refer to Section 12.0 for details.

**Note:** A number of these tests will fail. See Section 2.7 for further information.

**Note:** Open vSwitch builds with a number of warnings (of type deprecated declaration) originating from the original Open Source Open vSwitch v1.5.0 release package.

**Note:** It may be necessary to create a number of directories to ensure correct operation of the vswitch daemon:

```bash
mkdir -p /usr/local/etc/openvswitch
mkdir -p /usr/local/var/run/openvswitch
```
4.3 Compile QEMU

Once Open vSwitch is built, perform the following steps to build QEMU:

```bash
cd qemu
./configure --enable-kvm --dpdkdir=/path/to/dpdk --target-list=x86_64-softmmu
make
```

§ §
5.0 Compiling and Running Sample Applications

5.1 VirtIO

This section contains instructions on how to compile and run a sample application that demonstrates performance of an Intel® DPDK-accelerated version of VirtIO for IO virtualization.

5.1.1 Host Setup

5.1.1.1 Configure Kernel Boot Parameters

Start the OS with the following added kernel boot options. This ensures that a single hugepage, sized 1 GB, is available:

default_hugepagesz=1G hugepagesz=1G hugepages=1

5.1.1.2 Build Source Code

Compile Intel® DPDK, Open vSwitch, and QEMU as described in Section 4.0.

5.1.1.3 Setup Intel® DPDK

Once compilation of the above packages is complete, mount the Intel® DPDK hugepage:

`mount -t hugetlbfs nodev /mnt/huge`

Ensure that this is the only `hugetlbfs` mount point, by verifying a single entry for `hugetlbfs`, as output by the `mount` command.

`mount | grep huge`

The output of this command should be:

`nodev on /mnt/huge type hugetlbfs (rw, realtime)`

In the event that `hugetlbfs` is mounted on `/dev/hugepages` (or any other mountpoint, other than `/mnt/huge`), unmount this entry (`umount /dev/hugepages`), and remount `/mnt/huge`, as previously described.

5.1.1.4 Add Ports to the vSwitch

Add flow ports to the switch for any VirtIO and/or physical devices required, using `ovs-vsctl`. See Section 9.4 for details on how to add ports to the switch and Section 13.0 for details about the valid values for VirtIO ports.
### 5.1.1.5 Start Intel® DPDK vSwitch (ovs_dpdk)

Start the ovs_dpdk application. For example:

```
sudo ./openvswitch/datapath/dpdk/build/ovs_dpdk -c 0xf -n 4 --proc-type=primary --
huge-dir /mnt/huge -- -p 0x03 -k 2 -n 4 --stats=1 --vswitchd=0 --
client_switching_core=1 --config="(0,0,2),(1,0,3)"
```

**Note:** Intel® DPDK v1.5.2 does not automatically bind the igb uio driver to supported NICs. To manually bind a NIC to the Intel® DPDK driver, use the `pci unbind.py` script in `$RTE_SDK/tools/`. Consult the Intel® Data Plane Development Kit (Intel® DPDK) - Release 1.5.2 - Documentation for details.

**Note:** ovs_dpdk should only be started after ovs-vsctl, add-br, and add-port operations have completed.

### 5.1.1.6 Start QEMU

**Note:** QEMU will fail if ovs_dpdk is not already running.

```
sudo ./qemu/x86_64-softmmu/qemu-system-x86_64 -c <core_mask> -n <num> --proc-type secondary -- -cpu host -boot c -hda <path_to_image> -m <mem> -netdev dpdk,port=<port_num>,id=<device_id> -device virtio-net-pci,netdev=<device_id>,mac=<device_mac> -smp <smp> --enable-kvm -name "<client_name>" -nographic -vnc :<vnc_num>
```

Sample command line:

```
sudo ./qemu/x86_64-softmmu/qemu-system-x86_64 -c <core_mask> -n <num> --proc-type=secondary - -cpu host -boot c -hda <PATH_TO_IMAGE>.qcow2 -m 512 -netdev dpdk,port=1,id=me1 -device virtio-net-pci,netdev=me1,mac=00:00:00:00:00:01 -smp 2 --enable-kvm -name "Client 1" -nographic -vnc :1
```

**Note:** This command will start the guest image in persistent mode, i.e., all changes made in the guest remain present across reboots. Alternatively, the guest may be started in snapshot mode by passing the `-snapshot` flag on the command line:

```
sudo ./qemu/x86_64-softmmu/qemu-system-x86_64 -c <core_mask> -n <num> --proc-type=secondary - -snapshot -cpu host -boot c -hda <PATH_TO_IMAGE>.qcow2 -m 512 -netdev dpdk,port=1,id=me1 -device virtio-net-pci,netdev=me1,mac=00:00:00:00:00:01 -smp 2 --enable-kvm -name "Client 1" -nographic -vnc :1
```

### 5.1.1.7 Program the Switch’s Flow Tables

The switch’s flow table must be populated to allow traffic to flow to and from a VM, via the switch. See Section 9.0 and Section 10.0 for more information on programming flow tables, and Section 5.6 for an example of how to program the switch’s flow tables.

Add flows to switch traffic appropriately:

- From ingress port to VirtIO port/ring used by VM (if using an external traffic generator)
- From VirtIO port to next target (Physical/VirtIO/KNI/IVSHM port)
- Any additional entries required to complete the datapath

**Note:** The ID of the VirtIO port should be a value from 1 to 15.
5.1.2 **Guest Setup**

*Note:* The following configuration must be performed on each VirtIO client.

5.1.2.1 **Ensure VirtIO Ethernet Device Is Present**

After logging on to the client(s), list the PCI devices available and look for the entry listed as “Ethernet Controller.” This is the VirtIO device that has been mapped from the host:

```
lspci
```

The expected output should be:

```
00:03.0 Ethernet Controller: Red Hat, Inc Virtio network device
```

5.1.2.2 **Configure VM Network Interfaces**

*Note:* To pass packets correctly between VMs, the flow table must be configured correctly.

*Note:* A static ARP entry may be required depending on how the flow table has been configured.

The device can be configured like a standard Ethernet device.

Sample command line:

```
ifconfig eth0 up
ifconfig eth0.700 2.2.2.1/24 up
arp -s 2.2.2.2 00:00:00:00:00:02
```

5.2 **IVSHM Setup**

Intel® DPDK vSwitch supports the mapping of a host-created Intel® DPDK hugepage directly into guest userspace, eliminating performance penalties presented by QEMU I/O emulation.

This section contains instructions on how to compile and run a sample application that demonstrates performance of Intel® DPDK vSwitch with IVSHM integration. It also describes the additional configuration required for both host and client systems to use IVSHM.

*Note:* IVSHM modifications require QEMU v1.4.0 or above. Use of the IVSHM model and older versions of QEMU has not been validated.

*Note:* The current IVSHM implementation may present security issues in a multi-VM environment. Please refer to Section 2.0 for details.

5.2.1 **Host Setup**

5.2.1.1 **Configure Kernel Boot Parameters**

Start the OS with the following added kernel boot options. This ensures that a single hugepage, sized 1 GB, is used:

```
default_hugepagesz=1G hugepagesz=1G hugepages=1
```
5.2.1.2 Build Source Code

Compile Intel® DPDK, Open vSwitch, and QEMU as described in Section 4.0.

5.2.1.3 Set Up the Intel® DPDK

Once compilation of the above packages is complete, insert the Intel® DPDK kernel module and mount the hugepage. This will be mapped to the guests.

modprobe uio
insmod $RTE_SDK/$RTE_TARGET/kmod/igb_uio
mount -t hugetlbfs nodev /mnt/huge

Ensure that this is the only hugetlbfs mount point, by verifying a single entry for hugetlbfs, as output by the mount command.

mount | grep huge

The output of this command should be:

nodev on /mnt/huge type hugetlbfs (rw, realtime)

In the event that hugetlbfs is mounted on /dev/hugepages (or any other mountpoint, other than /mnt/huge), unmount this entry (umount /dev/hugepages), and remount /mnt/huge, as previously described.

See Section 9.0 and Section 10.0 for details on how to program the switch’s flow tables, and Section 5.4 for an example of a test setup.

5.2.1.4 Add Ports to the vSwitch

Add flow ports to the switch for any IVSHM and/or physical devices required, using ovs-vsctl. See Section 8.0 for details on how to add ports to the switch and Section 13.0 for details on the valid values for IVSHM ports.

5.2.1.5 Start Intel® DPDK vSwitch (ovs_dpdk)

Start the ovs_dpdk application.

Note: Intel® DPDK v1.5.2 does not automatically bind the igb_uio driver to supported NICs. To manually bind a NIC to the Intel® DPDK driver, use the pci_unbind.py script in $RTE_SDK/tools/. Consult the Intel® Data Plane Development Kit (Intel® DPDK) - Release 1.5.2 - Documentation for details.

sudo ./openvswitch/datapath/dpdk/build/ovs_dpdk -c <core_mask> -n 4 --proc-type-primary --huge-dir /mnt/huge --p <port_mask> -n <number_of_clients> -k 2 --stats=<stats update interval> --vswitchd=<core_mask> --client_switching_core=<core_mask> --config="<port_config>"

Sample command line:

sudo ./openvswitch/datapath/dpdk/build/ovs_dpdk -c 0xF -n 4 --proc-type-primary --huge-dir /mnt/huge --p 0x3 -n 3 -k 2 --stats=1 --vswitchd=0 --client_switching_core=1 --config="(0,0,2),(1,0,3)"

Note: Client 0 represents the vswitchd interface, and is always counted toward the number of clients present, i.e., to support two VMs, a value of 3 should be used as the number_of_clients parameter.

Note: ovs_dpdk should only be started after ovs-vsctl, add-br, and add-port operations have completed.
5.2.1.6 Program the Switch’s Flow Tables

The switch’s flow table must be populated to allow traffic to flow to and from a VM, via the switch and IVSHM rings. See Section 9.0 and Section 10.0 for more information on programming flow tables, and see Section 5.5 for an example of how to program the switch’s flow tables.

Add flows to switch traffic appropriately:

- From ingress port to IVSHM port/ring used by VM
- From IVSHM port to next target (Physical/Virtio/KNI/IVSHM port)
- Any additional entries required to complete the datapath

**Note:** The ID of the IVSHM port is the same as the `client_id` passed to the `ovs_client` application described in Section 5.2.2.6, and it should be a value from 1 to 15.

5.2.1.7 Copy Required Files to a Temporary Location

The `ovs_client` source code, Intel® DPDK source code, and Intel® DPDK runtime mapping information must be copied to each guest required. The simplest way to do this is by copying the required files to a directory on the host and mounting this directory as a drive on the guest. Once the guest is started, the files can be copied from the mounted drive to a local directory. This method has been validated using `qcow2` images.

```bash
mkdir /tmp/share
mkdir /tmp/share/DPDK
chmod 777 /tmp/share
cp -a /path/to/ovs_client/* /tmp/share
cp -a /path/to/DPDK/* /tmp/share/DPDK
cp -a /var/run/.rte_* /tmp/share
```

5.2.1.8 Start QEMU

Start QEMU on the host:

```
./qemu/x86_64-softmmu/qemu-system-x86_64 -c <coremask> -n 4 --proc-type=secondary
-cpu host -smp 2 -hda <path_to_guest_image> -m 4096 -boot menu=on -vnc
:/<vnc_session_id> --enable-kvm -device ivshmem,size=1024,shm=fd:/mnt/huge/rtemap_0
-drive file=fat:/tmp/share
```

**Note:** This will start the guest image in persistent mode, i.e., all changes made in the guest remain present across reboots. The guest may alternatively be started in snapshot mode by passing the `-snapshot` flag on the command line, and appending `-snapshot=off` to the `-drive parameter:

```
./qemu/x86_64-softmmu/qemu-system-x86_64 -c <coremask> -n 4 --proc-type=secondary
--snapshot -cpu host -smp 2 -hda <path_to_guest_image> -m 4096 -boot menu=on -vnc
:/<vnc_session_id> --enable-kvm -device ivshmem,size=1024,shm=fd:/mnt/huge/rtemap_0
-drive file=fat:/tmp/share,snapshot=off
```

5.2.2 Guest Configuration

**Note:** The following configuration must be performed on each IVSHM client.
5.2.2.1 **Enable Hugepages**

Start the guest OS with the following added kernel options. This ensures that hugepages are enabled in the guest thus allowing for correct operation of the ovs_client application:

```
default_hugepagesz=2M hugepagesz=2M hugepages=1024
```

5.2.2.2 **Obtain PCI Device ID of Mapped Memory**

After logging on to the client, list the PCI devices available and look for the entry listed as “Ram Memory.” This is the hugepage that has been mapped from the host:

```
lspci
```

The expected output should be:

```
00:04.0 RAM Memory: Red Hat, Inc Device 1110
```

Make note of the PCI device ID and verify that this device path is present in the client:

```
ls /sys/devices/pci0000:00/0000:00:04:0/resource2
```

5.2.2.3 **Link Intel® DPDK Hugepage in the Guest**

Create a symbolic link in the guest to use the mapped hugepage instead of the standard local hugepage file. Point /mnt/huge/rtemap_0 (default mount point of the local hugepage, as specified by the ovs_dpdk application) to the location of the PCI device bar obtained in the previous step:

```
l –s /sys/devices/pci0000:00/0000:00:04:0/resource2 /mnt/huge/rtemap_0
```

**Note:** The local hugepage must not be mounted to /mnt/huge; instead it must be mounted to a different area.

Compile Intel® DPDK in the guest, and mount hugepages to a non-standard area. There is no need to insert the igb_uio kernel module:

```
mkdir /mnt/hugepages
mount –t hugetlbfs nodev /mnt/hugepages
```

5.2.2.4 **Copy Required Files from Host**

In the guest, mount the temporary folder, which was created in the host, and copy the required files:

```
mkdir -p /mnt/ovs_client
mkdir -p /root/ovs_client
mount –o iocharset=utf8 /dev/sdb1 /mnt/ovs_client
cp –a /mnt/ovs_client/.rte_* /var/run
cp –a /mnt/ovs_client/* /root/ovs_client
```

5.2.2.5 **Compile Intel® DPDK**

```
cd /root/ovs_client/DPDK
export RTE_SDK=/root/ovs_client/DPDK
export RTE_TARGET=x86_64-default-linuxapp-gcc
make install T=x86_64-default-linuxapp-gcc
```
5.2.2.6 Compile and Run ovs_client Sample Application

```
cd /root/ovs_client
make
./build/ovs_client -c <core_mask> -n 4 --proc-type=secondary -- -n <client_id>
```

5.2.3 Optimizing Performance

To maximize throughput, assign individual cores to each of the various processes involved in the test setup (either using the `taskset` command, or the core mask passed to the `ovs_client` and `ovs_dpdk` applications). Additionally, on the host, all available cores, with the exception of core 0, should be isolated from the kernel scheduler.

5.2.3.1 Sample Setup for 8-core System (16 Logical Cores if Intel® Hyper-Threading Technology Enabled)

5.2.3.1.1 Isolate Cores

In the host, edit `/boot/grub2/grub.cfg` (or `/etc/default/grub`, if applicable), specifically this line:

```
GRUBCMDLINELINUX="...
```

Include the following:

```
isolcpus=1,2,...,n
```

*Note:* `n` should be the max number of logical cores available (if Intel® Hyper-Threading Technology is enabled). Always leave core 0 for the operating system.

Update the grub configuration file.

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

Reboot the system, then set up the application as described above but with two differences.

First, for `ovs_dpdk`, substitute in this command:

```
sudo ovs_dpdk -c 0x0F -n 4 --proc-type=primary -- -n 3 -p 0x3 -k 2 --stats=1 --vswitchd=0 -client_switching_core=1 --config="(0,0,2),(1,0,3)"
```

Then for `ovs_client`, substitute in this command:

```
ovo_client -c 0x1 -n 4 --proc-type-secondary -- -n 1
ovo_client -c 0x1 -n 4 --proc-type-secondary -- -n 2
```

You can use Table 2 to guide you in affinitizing the host core.

*Note:* For all Intel® DPDK-enabled applications, the core mask option (`-c`) must be set so that no two processes have overlapping core masks.
5.3 Intel® DPDK KNI Setup

When created in a guest, KNI devices enable non-Intel® DPDK applications running in the VM to use the Intel® DPDK shared hugepage using the IVSHM model.

This section contains instructions on how to compile and run a sample application that, when run in the guest, allows the user to create an Intel® DPDK KNI device, which will attach to queues in the host ovSwitch application. It also describes the additional configuration required for both host and client systems to use KNI.

**Note:** This release supports only the KNI implementation contained within Intel® DPDK v1.5.2

### 5.3.1 Host Configuration

Follow the host configuration steps provided in [Section 5.2.1](#).

**Note:** When programming the Switch’s flow table, the IVSHM port value should be replaced with a KNI FIFO value (in the range 32-47; i.e., 32 = KNI FIFO 0, 33 = KNI FIFO 1, and so on).

Then copy the KNI patch file to a temporary location:

```bash
cp /path/to/kni/kni_misc.c /tmp/share
```

### 5.3.2 Guest Configuration

Follow the guest configuration steps, as described in [Section 5.2.2](#), up until the “Compile and Run ovs_client Sample Application” step.

### 5.3.2.1 Insert the rte_kni Module

A small number of modifications to the standard Intel® DPDK driver are required to support KNI devices in the guest. These changes have been included as a patch. Apply the *kni_misc* patch before compiling Intel® DPDK and inserting the KNI module.

```bash
cd DPDK
patch -n ./DPDK/lib/librte_eal/linuxapp/kni/kni_misc.c < kni_misc.patch
make install T=x86_64-default-linuxapp-gcc
insmod ./x86_64-default-linuxapp-gcc/kmod/rte_kni.ko
```

### Table 2. Affinitizing the Host Core

<table>
<thead>
<tr>
<th>Process</th>
<th>Core</th>
<th>Core Mask</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>0</td>
<td>0x1</td>
<td>All other CPUs isolated (isolcpus boot parameter)</td>
</tr>
<tr>
<td>client_switching_core</td>
<td>1</td>
<td>0x2</td>
<td>Affinity set in ovs_dpdk command line</td>
</tr>
<tr>
<td>RX core</td>
<td>2</td>
<td>0x4</td>
<td>Affinity set in ovs_dpdk command line</td>
</tr>
<tr>
<td>RX core</td>
<td>3</td>
<td>0x8</td>
<td>Affinity set in ovs_dpdk command line</td>
</tr>
<tr>
<td>QEMU process VM1</td>
<td>4</td>
<td>0x10</td>
<td><code>taskset -a &lt;pid_of qemu_process&gt;</code></td>
</tr>
<tr>
<td>QEMU process VM1</td>
<td>5</td>
<td>0x20</td>
<td><code>taskset -a &lt;pid_of qemu_process&gt;</code></td>
</tr>
<tr>
<td>QEMU process VM2</td>
<td>6</td>
<td>0x40</td>
<td><code>taskset -a &lt;pid_of qemu_process&gt;</code></td>
</tr>
<tr>
<td>QEMU process VM2</td>
<td>7</td>
<td>0x80</td>
<td><code>taskset -a &lt;pid_of qemu_process&gt;</code></td>
</tr>
</tbody>
</table>
5.3.2.2 Compile and Run kni_client Sample Application

Copy the kni_client folder to a directory on the VM, then compile and run it. When the application is running, bring up the KNI device.

cd kni_client
make
./build/kni_client -c 0x1 --proc-type-secondary -- -p <kni_portmask> &
ifconfig vEthX up  #where X is the number of a KNI devices configured in the portmask

Note: kni_portmask above is similar to the ovś_dpdk portmask. Refer to Section 7.0 for details. However, the kni_portmask should be entered in decimal format only (i.e., no prepending 0x).

5.4 Sample Test Setup (Physical Port-Physical Port)

To forward packets along the various datapaths supported by ovś_dpdk (physical port to physical port, physical port to VM, VM to physical port, VM to VM), the switch's flow table must be programmed, either directly using ovś-dpctl, or indirectly using ovś-ofctl. This section describes how to program the flow table to forward a specific flow from one physical port to another physical port via the switch (Phy Port -> Switch -> Phy Port), using ovś-dpctl.

5.4.1 Test Setup

Perform the initial setup, as described in the section of the relevant I/O method, and Section 10.0.

This test assumes that ingress traffic is received on Physical Port 0, either via a traffic generator or otherwise.

5.4.2 Add a Flow Entry

sudo ./ovś-dpctl -c 1 --proc-type-secondary -- -s add-flow dpdk@dp
"in_port(16),eth(src=00:00:00:00:00:11,dst=00:00:00:00:00:22),eth_type(0x0800),ipv4(src=1.1.1.1,dst=1.1.1.2,proto=1,tos=0,ttl=64,frag=no)" "17"

This command adds an entry in the switch's flow table that sends packets received on physical port 0 with source MAC address 00:00:00:00:00:11, destination MAC address 00:00:00:00:00:22, Ethertype 0x800 (IPv4), source IP address 1.1.1.1, destination IP address 1.1.1.2, protocol type ICMP, Type Of Service 0 and Time To Live 64, to physical port 1.

When matching traffic is received on physical port 0, it should be switched to the correct physical port. Enable statistics for ovś_dpdk (specify non-zero value for --stats parameter) to observe traffic passing through the various interfaces. This can also be useful for debugging.

Note: When using dpctl, the flow must be specified exactly (i.e., all 13 match fields must be specified). If a match field is not specified, ovś_dpdk will assume that it is zero.

If bidirectional testing is required, another flow entry may be added to switch packets received on physical port 1 to physical port 0:

sudo ./ovś-dpctl -c 1 --proc-type-secondary -- -s add-flow dpdk@dp
"in_port(17),eth(src=00:00:00:00:00:22,dst=00:00:00:00:00:11),eth_type(0x0800),ipv4(src=1.1.1.2,dst=1.1.1.1,proto=1,tos=0,ttl=64,frag=no)" "16"

Note: When using dpctl, the ovś-vswitchd application should not be running.
5.5 Sample Test Setup (Physical Port to VM to Physical Port via IVSHMEM)

This section describes how to program the flow table to forward a specific flow from one physical port to a VM and back out to another physical port via the switch (Phy Port -> Switch -> VM -> Switch -> Phy Port), using ovs-dpctl.

5.5.1 Test Setup

Perform the initial setup, as described in the IVSHM section, and Section 10.0.

This test assumes that ingress traffic is received on Physical Port 0, either via a traffic generator or otherwise, and that the VM/ovs_client app uses IVSHM ring 1.

5.5.2 Add a Flow Entry

```
sudo ./ovs-dpctl -c 1 --proc-type=secondary -- -s add-flow dpdk8dp "in_port(16),eth(src=00:00:00:00:00:11,dst=00:00:00:00:00:22),eth_type(0x0800),ipv4(src=1.1.1.1,dst=2.2.2.1,proto=1,tos=0,ttl=64,frag=no)" "1"
```

This command adds an entry in the switch's flow table that sends packets received on physical port 0 with source MAC address 00:00:00:00:00:11, destination MAC address 00:00:00:00:00:22, Ethertype 0x800 (IPv4), source IP address 1.1.1.1, destination IP address 2.2.2.1, protocol type ICMP, Type Of Service 0 and Time To Live 64, to the VM.

When matching traffic is received on physical port 0, it should be switched to the VM. Enable statistics for ovs_dpdk (specify non-zero value for -stats parameter) to observe traffic passing through the various interfaces. This can also be useful for debugging.

**Note:** When using dpctl, the flow must be specified exactly (i.e., all 13 match fields must be specified). If a match field is not specified, ovs_dpdk will assume that it is zero.

To allow traffic from ovs_client running on the VM to pass traffic out, another flow entry may be added to switch packets received from VM to physical port 1:

```
sudo ./ovs-dpctl -c 1 --proc-type=secondary -- -s add-flow dpdk8dp "in_port(1),eth(src=00:00:00:00:00:11,dst=00:00:00:00:00:22),eth_type(0x0800),ipv4(src=1.1.1.1,dst=2.2.2.1,proto=1,tos=0,ttl=64,frag=no)" "16"
```

**Note:** When using dpctl, the ovs-vswitchd application should not be running.

5.6 Sample Test Setup (VM to VM via VirtIO)

This section describes how to program the flow table to forward a specific flow between VM to VM via the switch (VM -> Switch -> VM), using ovs-ofctl.

5.6.1 Test Setup

Perform the initial setup, as described in the VirtIO section, Section 5.1, and Section 10.0, Dynamic Flow Manipulation Using ovs-ofctl.

This test assumes that traffic originates from VM0 and that VMs 0 and 1 use VirtIO devices/rings 1 and 2, respectively.
5.6.2 Add a Flow Entry

```
sudo ./utilities/ovs-ofctl add-flow br0
in_port=1,dl_type=0x0800,nw_src=2.2.2.1,nw_dst=2.2.2.2,idle_timeout=0,action=output:2
```

This command adds an entry in the switch’s flow table that sends packets sent by VM 0, with any source and destination MAC address, Ethertype 0x800 (IPv4), source IP address 2.2.2.1, destination IP address 2.2.2.2, to VM 1.

When matching traffic is received by the switch from VM 0, it should be switched to the correct VM port. Enable statistics for `ovs_dpdk` (specify non-zero value for `-stats` parameter) to observe traffic passing through the various interfaces. This process can also be useful for debugging.

**Note:** When using `ofctl`, the flow will match wildcard fields, which are not configured by the user. If a match field is not specified, `ovs_dpdk` will assume that it is wildcarded.

To allow traffic from `ovs_client` running on the VM to pass traffic out, another flow entry may be added to switch packets received from VM 1 to physical port 1:

```
sudo ./utilities/ovs-ofctl add-flow br0
in_port=2,dl_type=0x0800,nw_src=2.2.2.2,nw_dst=2.2.2.1,idle_timeout=0,action=output:16
```

5.6.2.1 Configure VM Network Interfaces

**Note:** To pass packets correctly between VMs, the flow table must be configured correctly.

**Note:** A static ARP entry may not be required, depending on how the flow table has been configured.

The device can be configured as a standard Ethernet device.

Sample command line:

```
ifconfig eth0 up
ifconfig eth0.700 2.2.2.1/24 up
arp -s 2.2.2.2 00:00:00:00:00:02
```
6.0 Wind River* Linux* 5.0.1 with Wind River* Linux* Open Virtualization Profile

This section provides the steps for using Wind River* Linux* Host and Wind River* Linux* Guest with Intel® DPDK vSwitch. For additional information, please refer to:

- Wind River Linux 5.0.1 User’s Guide
- Wind River Linux 5.0.1 Open Virtualization Profile: Virtual Node User’s Guide

6.1 Building the Host and Guest

The following subsections show the basic configuration steps required to build the Wind River* Linux* Host and Guest for an Intel® Xeon® processor-based platform. Once complete, a kernel and rootfs will be available for both host and guest deployment on the target system.

6.1.1 Getting Ready to Build

The host and guest build steps will reference the following environment variables.

- PROD_BASE - This is the directory where Wind River* Linux* is installed. Typically something like /opt/Windriver.
- WIND_BASE - This is the base Linux* directory in the Wind River* Linux* product installation.
- HOST_BUILD_DIR - is the directory in which you will build the Wind River* Linux* host.
- GUEST_BUILD_DIR - is the directory in which you will build the Wind River* Linux* guest.

1. Set up the custom variables and create the build directories.

   $ export PROD_BASE=/opt/Windriver
   $ export HOST_BUILD_DIR=~/build/host
   $ export GUEST_BUILD_DIR=~/build/guest
   $ mkdir -p $HOST_BUILD_DIR
   $ mkdir -p $GUEST_BUILD_DIR

2. Set up the build environment:

   This step will set up the SWIND_BASE environment variable along with others to enable building the host and guest.

   $ $PROD_BASE/wrenv.sh -p -wrlinux-5
6.1.2 Wind River* Linux* Host Build

The following steps provide the framework for building the host.

1. Configure the host build with the following options:

   $ cd $HOST_BUILD_DIR
   $ $WIND_BASE/wrlinux/configure
       --enable-board=intel-xeon-core
       --enable-kernel=preempt-rt
       --enable-rootfs=ovp-kvm
       --enable-addons=wr-ovp
       --with-layer=wr-intel-support
       --with-template=feature/openvswitch,feature/rt-tune
       --enable-reconfig
       --enable-parallel-pkgbuilds=8 --enable-jobs=8

   Note: --enable-parallel-pkgbuilds=8 and --enable-jobs=8 can be tuned based on the processing power of the build platform. (See $GUEST_BUILD_DIR/local.conf.)

2. Extract and patch the kernel source for kernel configuration:

   $ make -C build linux-windriver.config

3. Launch the menu-based tool for guest kernel configuration. Here you will add the appropriate kernel options for your hardware.

   $ make -C build linux-windriver.menuconfig

4. Regenerate the kernel configuration

   $ make -C build linux-windriver.rebuild

5. Build a new file system

   $ make
6.1.3 Wind River* Linux* Guest Build

The following steps provide the framework for building the guest.

1. Configure the guest build:

```
$ cd $GUEST_BUILD_DIR
$ $WIND_BASE/wrlinux/configure \
   --enable-board=x86-64-kvm-guest \
   --enable-kernel=preempt-rt \
   --enable-rootfs-ovp-guest+kvm \
   --enable-addons=wr-ovp \
   --enable-reconfig \ 
   --enable-parallel-pkgbuilds=8 \ 
   --enable-jobs=8
```

*Note:* --enable-parallel-pkgbuilds=8 and --enable-jobs=8 can be tuned based on the processing power of the build platform. (See $GUEST_BUILD_DIR/local.conf.).

2. Extract and patch the kernel source for kernel configuration:

```
$ make -C build linux-windriver.config
```

3. Launch the menu-based tool for kernel configuration. Here you will add guest kernel options. Turn on these options: CONFIG_PREEMPT_NONE=y and CONFIG_BLK_DEV_PIIX=y through the kernel configuration menu.

```
$ make -C build linux-windriver.menuconfig
```

4. Regenerate the kernel configuration to add the appropriate kernel options for the hardware.

```
$ make -C build linux-windriver.rebuild
```

5. Build a new file system:

```
$ make
```

```
```
7.0 Intel® DPDK vSwitch Command Line Parameters

This section explains the various command-line parameters passed to the Intel® DPDK vSwitch application.

Sample command line:

```
sudo ./datapath/dpdk/build/ovs_dpdk -c 0x0F -n 4 --proc-type=primary --huge-dir /mnt/huge -- -p 0x03 -n 4 -k 2 --stats=1 --vswitchd=0 --client_switching_core=1 --config="(0,0,2),(1,0,3)"
```

Note: The initial parameters, before the separating double-dash ("--"), are Intel® DPDK-specific options, details of which can be found in the Intel® DPDK Getting Started Guide.

The Intel® DPDK vSwitch application-specific options are detailed below:

- `--stats`: If zero, statistics are not displayed. If nonzero, it represents the interval in seconds at which statistics are updated onscreen.
- `--client_switching_core`: CPU ID of the core on which the main switching loop will run.
- `-n NUM`: The number of supported clients.
- `-p PORTMASK`: Hexadecimal bitmask representing the ports to be configured, where each bit represents a port ID, i.e., for a portmask of 0x3, ports 0 and 1 are configured.
- `-k KNIPORTMASK`: Number of KNI devices to configure.

Note: Currently, this parameter must be used in all use cases, not just KNI.

- `-v NUM`: Number of virtual Ethernet devices to configure.

Note: This parameter is optional, and is typically used only when running OFTest. The maximum number of virtual devices currently supported is 8.

- `--vswitchd`: CPU ID of the core used to display statistics and communicate with the vswitch daemon.
- `--config (port,queue,lcore)],(port,queue,lcore]`: Each port/queue/core group specifies the CPU ID of the core that will handle ingress traffic for the specified queue on the specified port.
8.0 Switch Manipulation Using ovs-vsctl

8.1 ovs-vsctl

Intel® DPDK vSwitch supports dynamic switch manipulation, using the ovs-vsctl command. A subset of the command’s operations has been implemented, as described later in this section. An example of ovs-vsctl usage is described in Section 9.4.

ovs-vsctl syntax is generally in the format:

ovs-vsctl COMMAND BRIDGE [PORT] – [OPTIONS]

Where:

- COMMAND: One of the supported commands described in Section 8.2
- BRIDGE: The bridge name, e.g., br0
- PORT: The port name, e.g., ovs_dpdk_16
- OPTIONS: The options for the switch. Currently, one of the following:
  - set Bridge datapath_type=TYPE
  - set Interface type=TYPE
- TYPE: The bridge type, e.g., dpdk

Note: The --no-wait option should be used in cases where the daemon is not running.

8.2 Supported Commands

8.2.1 add-br

Create new bridge named BRIDGE:

add-br BRIDGE

Example:

# sudo ./ovs-vsctl add-br br0 -- set Bridge br0 datapath_type=dpdk

8.2.2 del-br

Delete bridge named BRIDGE:

del-br BRIDGE

Example:

# sudo ./ovs-vsctl del-br br0
8.2.3 add-port

Add new PORT to BRIDGE:

add-port BRIDGE PORT

Example:

# sudo ./ovs-vsctl add-port br0 ovs_dpdk_16 --set Interface type=dpdk

8.2.4 del-port

Delete PORT from BRIDGE:

del-port BRIDGE PORT

Example:

# sudo ./ovs-vsctl del-port br0 ovs_dpdk_16

§ §
9.0 Dynamic Flow Manipulation Using ovs-ofctl

9.1 ovs-ofctl

Intel® DPDK vSwitch supports dynamic flow manipulation, using the ovs-ofctl command. A subset of the command’s operations has been implemented, as described later in this section.

ovs-ofctl syntax is generally in the format:

```
ovo-ofctl COMMAND SWITCH FLOW
```

Where:
- SWITCH: The switch name, e.g., br0
- COMMAND: One of the supported commands described in Section 9.2
- FLOW: A comma-separated list of the strings described in Section 9.3

Note: Matching on VLAN header fields (dl_vlan, dl_vlan_pcp) is supported as of release 0.6.0. This has been validated for the phy-phy case only.

Note: A timeout value may also be specified for a flow. When this value expires, an entry is deleted from the vswitch daemon’s flow table:

```
idle_timeout=timeout
```

Note: A flow may only specify a value for an L3 field if it also specifies a particular L2 protocol. Similarly, a flow may only specify an L4 field if it also specifies particular L2 and L3 protocol types.

9.2 Supported Commands

The following commands are currently supported.
9.2.1 add-flow

Add a flow described by FLOW to SWITCH:

```
add-flow SWITCH FLOW
```

**Note:** add-flow requires an additional FLOW parameter: action=ACTION. Currently, the only supported actions are those listed in Section 2.6. For example, if using the OUTPUT action to output to <PORT>:

```
action=output:<PORT>
```

Example:

```
# ovs-ofctl add-flow br0 dl_type=0x0800,nw_src=10.0.124.4,nw_dst=10.0.124.1,
  idle_timeout=0,action=output:16
```

**Note:** As of release 0.7, multiple actions can be specified in the action field.

9.2.2 del-flows

Delete matching FLOWs from SWITCH. If FLOW is not set, all flows are deleted.

```
del-flows SWITCH [FLOW]
```

Example:

```
# ovs-ofctl del-flows br0
```

9.2.3 dump-flows

Print matching FLOWs. If FLOW is not set, prints all flows.

```
dump-flows SWITCH [FLOW]
```

Example:

```
# ovs-ofctl dump-flows br0
```
9.3 Supported Flow Strings

The following flow strings are currently supported:

<table>
<thead>
<tr>
<th>Command</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port=port</td>
<td>Datapath in port</td>
</tr>
<tr>
<td>dl_src=XX:XX:XX:XX:XX:XX</td>
<td>Source MAC</td>
</tr>
<tr>
<td>dl_dst=XX:XX:XX:XX:XX:XX</td>
<td>Destination MAC</td>
</tr>
<tr>
<td>dl_type=ethertype</td>
<td>Ethernet protocol type</td>
</tr>
<tr>
<td>dl_vlan=vlan</td>
<td>VLAN ID</td>
</tr>
<tr>
<td>dl_vlan_pcp=priority</td>
<td>VLAN Priority Code Point</td>
</tr>
<tr>
<td>nw_src=ip</td>
<td>Source IP</td>
</tr>
<tr>
<td>nw_dst=ip</td>
<td>Destination IP</td>
</tr>
<tr>
<td>nw_proto=proto</td>
<td>IP protocol type</td>
</tr>
<tr>
<td>nw_tos=tos</td>
<td>IP ToS</td>
</tr>
<tr>
<td>nw_ttl=ttl</td>
<td>Time to Live</td>
</tr>
<tr>
<td>tp_src=port</td>
<td>UDP or TCP source port</td>
</tr>
<tr>
<td>tp_dst=port</td>
<td>UDP or TCP destination port</td>
</tr>
</tbody>
</table>

9.4 Configuring Intel® DPDK vSwitch Flow Tables Using ovs-ofctl

To use ofctl, the vswitch daemon must be configured correctly.

Create the Open vSwitch database:

cd openvswitch
sudo ./ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db vswitchd/
vswitch.ovsschema

Start the Open vSwitch database server:

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

Configure the database:

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

Then add ports you want to use:

sudo ./utilities/ovs-vsctl --no-wait add-port br0 ovs_dpdk_16 -- set Interface
ovs_dpdk_16 type=dpdk
sudo ./utilities/ovs-vsctl --no-wait add-port br0 ovs_dpdk_17 -- set Interface
ovs_dpdk_17 type=dpdk

Note: In the example above, 16 and 17 refer to the port number index used in ovs_dpdk.
Ports 1-15 refer to VirtIO/IVSHM rings. Ports 16-31 refer to physical ports. Ports 32-47
refer to KNI FIFOs. Additional ports are required in the case of VirtIO/IVSHM/KNI.
You can see your configuration by typing the following:

```
sudo ./utilities/ovs-vsctl show
```

Start `ovs_dpdk`, as previously detailed. For example,

```
sudo ./datapath/dpdk/build/ovs_dpdk -c 0xf -n 4 -- -p 0xc -n 2 --stats=1 -- vswitchd=0 --client_switching_core=1 --config="(0,0,2),(1,0,3)"
```

Start the Open vSwitch daemon.

```
sudo ./vswitchd/ovs-vswitchd -c 0x100 --proc-type=secondary
```

**Note:** Before running the vSwitch daemon, ensure that the `ovs_dpdk` process has completed initialization.

Configure flow table entries using `ofctl`.

**Note:** By default, the daemon has a special flow entry implementing L2 learning, and every bridge has a default internal port with the same name as the bridge. Therefore, the daemon will add an output action on this default port causing a segmentation fault. To prevent this fault, you must first delete this flow table entry.

```
sudo ./utilities/ovs-ofctl del-flows br0
```

Then configure the flow you want to use:

```
sudo ./utilities/ovs-ofctl add-flow br0
<FLOW>,idle_timeout=0,action=<ACTION>
```

Where:

- `<ACTION>`: specifies the port to send the flow to, i.e., one of the actions listed in Section 2.6.
- `<FLOW>`: describes the flow match fields for the flow table entry. If a match field is not specified, then it as assumed to be a wildcard. The following fields are matched: `in_port`, `dl_src`, `dl_dst`, `dl_type`, `dl_vlan`, `dl_vlan_pcp`, `nw_src`, `nw_dst`, `nw_proto`, `nw_tos`, `nw_ttl`, `tp_src`, `tp_dst`. Refer to the `ovs-ofctl` man page for more details.

For example, the following command will send packets with ethertype 0x800, source IP address 10.0.124.4, and destination IP address 10.0.124.1 to physical port 0.

```
sudo ./utilities/ovs-ofctl add-flow br0
dl_type=0x0800,nw_src=10.0.124.4,nw_dst=10.0.124.1,idle_timeout=0,action=output:16
```
10.0 Dynamic Flow Manipulation Using ovs-dpctl

10.1 ovs-dpctl

Intel® DPDK vSwitch supports dynamic flow manipulation, using the ovs-dpctl command. A subset of the command’s operations has been implemented, as well as some new additional operations, as described later in this section. See the ovs-dpctl man page for further information.

ovs-dpctl syntax is generally in the format:

```
ovs-dpctl [DPDK_OPTIONS] -- [OPTIONS] COMMAND [ARG...]
```

Where:

- **DPDK_OPTION**: Specify options specific to the Intel® DPDK library. Currently:
  - `-c <core_mask>`
  - `-proc-type=secondary`
- **OPTIONS**: One of the options specified in Section 10.2
- **COMMAND**: One of the commands specified in Section 10.3
- **ARG**: One of the additional arguments, specified in Section 10.4, required by COMMAND

**Note:** tcp and udp cannot be used in the same flow.

**Note:** When using ovs-dpctl, the FLOW must be specified exactly (i.e., all 13 match fields must be specified). If a match field is not specified, ovs_dpdk will assume that it is 0.

**Note:** ovs-dpctl adds exact match flow table entries directly to the flow table of ovs_dpdk by sending messages via rings. In this way, flows can be added even when the vswitch daemon is not running, and it can be useful for debugging.

10.2 Supported Options

The following options are currently supported:
10.3 Supported Commands

The following commands are currently supported.

10.3.1 add-flow

Add FLOW with ACTIONS to DP:

```
add-flow DP FLOW ACTIONS
```

Example:

```
# sudo ./ovs-dpctl -c 1 -proc-type=secondary -- -s add-flow dpdk@dp
"in_port(16),eth)type(0x0800),ipv4(src=1.1.1.1,dst=1.1.1.2,proto=6,tos=0,ttl=64,fr
ag=no)" "17"
```

10.3.2 del-flow

Delete FLOW from DP:

```
del-flow DP FLOW
```

Example:

```
# sudo ./ovs-dpctl -c 1 -proc-type=secondary -- -s del-flow dpdk@dp
"in_port(16),eth)type(0x0800),ipv4(src=1.1.1.1,dst=1.1.1.2,proto=6,tos=0,ttl=64,fr
ag=no)" "17"
```

10.3.3 del-flows

Delete all flows from DP:

```
del-flows DP
```

Example:

```
# sudo ./ovs-dpctl -c 1 -proc-type=secondary -- -s del-flow dpdk@dp
```
10.3.4 **mod-flow**

Change FLOW actions to ACTIONS in DP:

```
mod-flow DP FLOW ACTIONS
```

Example:

```
# sudo ./ovs-dpctl -c 1 -proc-type=secondary -- -s mod-flow dpdk@dp
"in_port(16),eth)type(0x0800),ipv4(src=1.1.1.1,dst=1.1.1.2,proto=6,tos=0,ttl=64,frag=no)" "17" --clear
```

10.3.5 **get-flow**

Get FLOW actions from DP:

```
get-flow DP FLOW
```

Example:

```
# sudo ./ovs-dpctl -c 1 -proc-type=secondary -- -s get-flow dpdk@dp
"in_port(16),eth)type(0x0800),ipv4(src=1.1.1.1,dst=1.1.1.2,proto=6,tos=0,ttl=64,frag=no)" "17"
```

10.3.6 **dump-flows**

Display flows in DP:

```
dump-flows DP
```

Example:

```
# sudo ./ovs-dpctl -c 1 -proc-type=secondary -- -s dump-flows dpdk@dp
```
10.4 Supported Arguments

The following arguments are required by some commands:

Table 5. Arguments Required with ovs-dpctl

<table>
<thead>
<tr>
<th>Command</th>
<th>Comments</th>
</tr>
</thead>
</table>
| DP       | The datapath/switch name, in the format | datapath_type@datapath_name  
|          | Where datapath_type is dpdk and datapath_name is any string              |
| FLOW     | Any combination of the flow modifier‡ strings, separated by a comma      |
| ACTIONS  | Any combination of the supported actions**, separated by a comma         |

‡The following flow modifier strings are supported:

- `in_port(port_id)`
- `eth(src="XX;XX:XX:XX:XX",dst="YY:YY:YY:YY:YY")`
- `eth_type(ethertype)`
- `vlan(vlan)`
- `vlan_pcp(priority)`
- `ipv4(src="src_ip",dst="dst_ip",proto=ip_proto,tos=x,ttl=y,frag=z)`
- `tcp(src=src_port,dst=dst_port)`
- `udp(src=src_port,dst=dst_port)`

**For supported actions, please refer to Section 2.6.

§ §
11.0 Testing Intel® DPDK vSwitch with OFTest

OFTest is a test framework meant to exercise a candidate OpenFlow switch - in this case, Intel® DPDK vSwitch. It is especially useful when adding additional functionality to the switch or when modifying existing functionality. General information on the framework, along with the framework itself can be found here:

http://www.projectfloodlight.org/oftest/

This section contains instructions on how to configure and test Intel® DPDK vSwitch using OFTest.

11.1 Get OFtest

Clone the OFTest repo from GitHub:

```bash
git clone git://github.com/floodlight/oftest
```

The following section has been validated against a specific OFTest commit ID:

```bash
git checkout 624f55a79f774efdae0864715f0ca55ea6335aa6
```

OFTest requires a number of additional utilities to be installed prior to use- refer to the ‘Pre-requisites’ section of the Project Floodlight Getting Started Guide here:

http://docs.projectfloodlight.org/display/OFTest/Longer+Start

11.2 Configure Intel® DPDK vSwitch for OFTest

Follow the host configuration steps provided in Section 5.1.1, omitting the QEMU step. There are a small number of additional divergences from these steps, as outlined below.

11.2.1 Add Switch Ports

A number of ports should be added to the switch to connect with the virtual Ethernet (vEth) devices. The corresponding port numbers for vEth devices fall in the range 64-71.

11.2.2 Start Intel® DPDK vSwitch (ovs_dpdk)

OFTest requires a standard Linux networking interface in order to send traffic to the datapath of the switch via virtual Ethernet ports, i.e. vEthx. Support has been added to accommodate this requirement via Intel® DPDK’s host Kernel NIC Interface (KNI). This presents a standard Linux network interface to the host, which OFTest uses to transmit traffic to the switch.
Start the datapath with Host KNI enabled, by passing the -v parameter, as described in Section 7.0. For example:

```
sudo ./openvswitch/datapath/dpdk/build/ovs_dpdk -c 0xf -n 4 --proc-type=primary --
huge-dir /mnt/huge -- -p 0x03 -k 2 -n 4 -v 4 --stats=1 --vswitchd=0 --
client_switching_core=1 --config="(0,0,2),(1,0,3)"
```

**Note:** OFTest requires four (4) vEth devices

Additionally, these devices - which, by default, have names corresponding to vEthx - should be brought up. For example:

```
ifconfig vEth0 up
ifconfig vEth1 up
ifconfig vEth2 up
ifconfig vEth3 up
```

### 11.2.3 Configure the Switch

OFTest requires OpenFlow-controller connectivity to the switch. This, in turn, requires the `ovs-vswitchd` application be running and the switch configured to allow such a connection.

Follow Section 9.4 to configure Intel® DPDK vSwitch as required. However, before starting `ovs-vswitchd`, configure the switch to use an OpenFlow controller. For example:

```
./ovs-vsctl set-controller br0 tcp:127.0.0.1:6653
```

**Note:** Check the correct port number for the version of OFTest installed

Additionally, disable in-band management for the bridge. This should also be done before executing `ovs-vswitchd`. For example:

```
./ovs-vsctl set Bridge br0 other_config:disable-in-band=true
```

**Note:** When `ovs-vswitchd` starts, the switch will attempt to contact the controller. Until such time as OFTest (and thus, the controller) is run, a series of connection error messages (i.e. ‘connection failed’) are displayed - this is expected behavior, and may be disregarded.

### 11.3 Run OFTest

A full guide on how to execute OFTest can be found at the OFTest site (http://docs.projectfloodlight.org/display/OFTest/Getting+Started). In this case, OFTest should be configured to use the Host KNI devices created by `ovs_dpdk`. For example:

```
./oft -i 64@vEth0 -i 65@vEth1 -i 66@vEth2 -i 67@vEth3
```

**Note:** A number of these tests will fail. In most cases this is due to missing functionality in either standard Open vSwitch or Intel® DPDK vSwitch.

**Note:** If Python* has not been compiled with IPv6 support, Scapy will issue an error. To avoid this, either:

a. Recompile Python*, adding support for IPv6 by passing `--enable_ipv6` to the configure step,

or
b. Comment out any lines related to IPv6 in oftest/src/python/oftest/packet.py.

**Note:**
Intel® DPDK vSwitch complies with a subset of OpenFlow specification v1.0; by default, OFTest tests switch compatibility against v1.0 of the OpenFlow specification, but this can be specified explicitly using one of the command line options:

```
-V 1.0
```

or

```
--oft-version=1.0
```

A list of the expected results for OFTest follows (assuming no IPv6 testing):

```
load.PacketInLoad                OK
load.FlowModLoad                 OK
load.PacketOutLoad               OK
load.FlowRemovedLoad              ERROR
port_stats.AllPortStats           OK
port_stats.MultiFlowStats        FAIL
port_stats.SingleFlowStats       FAIL
pktact.ModifyL4Dst               FAIL
pktact.ModifyL2Src                FAIL
pktact.AllPlusIngress            OK
pktact.DirectArpPackets           OK
pktact.DirectTwoPorts             OK
pktact.WildcardPriorityWithDelete OK
pktact.DirectBadIpTcpPacketsBase  OK
pktact.Flood                      OK
pktact.ModifyL2DstIngressMC      FAIL
pktact.ModifyL4SrcUdp            FAIL
pktact.SingleWildcardMatchTagged  FAIL
pktact.DirectBadIpTcpPackets      OK, with warnings
pktact.SingleWildcardMatch        FAIL
pktact.StripVLANTagWithTagMatchWildcarded  FAIL
pktact.SingleWildcardMatchPriorityInsertModifyDelete OK
pktact.FloodPlusIngress          FAIL
pktact.WildcardPriority          FAIL
pktact.ModifyL4Src                FAIL
pktact.ModifyL2Dst                fail
pktact.AllExceptOneWildcardMatch FAIL
pktact.ModifyL2SrcDstMC           FAIL
pktact.AddVLANTag                 FAIL
pktact.DirectPacketControllerQueue(S) OK
pktact.ModifyL3Dst                FAIL
pktact.AllWildcardMatch           OK
pktact.DirectMC                    FAIL
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>pktact.DirectPacket</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.ModifyVID</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.FlowToggle</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.ModifyTOS</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.DirectMCNonIngress</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.AllWildcardMatchTagged</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.MatchEach</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.DirectBadPacketBase</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.ModifyVlanPcp</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.ModifyL2DstIngress</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.ModifyL2DstVIDMC</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.ModifyAll</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.BaseMatchCase</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.ModifyVIDWithTagMatchWildcarded</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.ModifyL4DstUdp</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.ModifyL2DstMC</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.FloodMinusPort</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.AllExceptOneWildcardMatchTagged</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.DirectVlanPacketsDoubleTagged</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.ExactMatchTagged</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.All</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.ModifyL2SrcMC</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.StripVLANTag</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.ModifyL3Src</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.DirectBadLlcPackets</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.SingleWildcardMatchPriority</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.DirectLlcPackets</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.ExactMatch</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.DirectPacketICMP</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.DirectPacketQueue</td>
<td>OK</td>
</tr>
<tr>
<td>pktact.ModifyVIDToIngress</td>
<td>FAIL</td>
</tr>
<tr>
<td>pktact.DirectVlanPacketsERROR</td>
<td>error</td>
</tr>
<tr>
<td>pktact.DirectBadIpUdpPackets</td>
<td>OK, with warnings</td>
</tr>
<tr>
<td>pktact.DirectPacketController</td>
<td>OK</td>
</tr>
<tr>
<td>openflow_protocol_messages.ModifyStateModify</td>
<td>OK</td>
</tr>
<tr>
<td>openflow_protocol_messages.ModifyStateDelete</td>
<td>OK</td>
</tr>
<tr>
<td>openflow_protocol_messages.ReadState</td>
<td>OK</td>
</tr>
<tr>
<td>openflow_protocol_messages.BarrierRequestReply</td>
<td>OK</td>
</tr>
<tr>
<td>openflow_protocol_messages.PacketOut</td>
<td>OK</td>
</tr>
<tr>
<td>openflow_protocol_messages.ConfigurationRequest</td>
<td>OK</td>
</tr>
<tr>
<td>openflow_protocol_messages.EchoWithoutBody</td>
<td>OK</td>
</tr>
<tr>
<td>openflow_protocol_messages.PacketIn</td>
<td>OK</td>
</tr>
<tr>
<td>openflow_protocol_messages.FeaturesRequest</td>
<td>OK</td>
</tr>
</tbody>
</table>
openflow_protocol_messages.ModifyStateAdd OK
openflow_protocol_messages.Hello OK
flow_expire.FlowExpire OK
actions.ModifyL4Dst FAIL
actions.Anouncement OK
actions.NoAction OK
actions.AddVlanTag OK
actions.ModifyL2Src FAIL
actions.ModifyTos FAIL
actions.ForwardLocal OK
actions.ForwardAll FAIL
actions.ModifyL4Src FAIL
actions.ForwardTable OK
actions.ForwardController FAIL
actions.ModifyL2Dst FAIL
actions.ForwardInport FAIL
actions.ModifyL3Dst FAIL
actions.ForwardFlood FAIL
actions.VlanPrio2 OK
actions.VlanPrio1 OK
actions.ModifyL3Src FAIL
actions.ModifyVlanTag OK
detailed_contr_sw_messages.NoOverlapChecking OK
detailed_contr_sw_messages.HardTimeout OK
detailed_contr_sw_messages.StrictVsNonstrict OK
detailed_contr_sw_messages.EmerFlowTimeout FAIL
detailed_contr_sw_messages.IdleTimeout OK
detailed_contr_sw_messages.OverlapChecking OK
detailed_contr_sw_messages.ModifyAction FAIL
detailed_contr_sw_messages.SendFlowRem OK
detailed_contr_sw_messages.FlowTimeout OK
detailed_contr_sw_messages.DeleteNonexistingFlow OK
detailed_contr_sw_messages.Outport2 FAIL
detailed_contr_sw_messages.Outport1 OK
detailed_contr_sw_messages.DeleteEmerFlow FAIL
detailed_contr_sw_messages.StrictModifyAction FAIL
detailed_contr_sw_messages.IdenticalFlows OK
detailed_contr_sw_messages.MissingModifyAdd OK
message_types.DescStatsReplyBody OK
message_types.QueueConfigReply FAIL
message_types.ErrorMsg FAIL
message_types.PortModFwd OK
message_types.SetConfigRequest OK
<table>
<thead>
<tr>
<th>Message Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>message_types.PacketInBodyAction</td>
<td>FAIL</td>
</tr>
<tr>
<td>message_types.PacketInBodyMiss</td>
<td>OK</td>
</tr>
<tr>
<td>message_types.PortModFlood</td>
<td>OK</td>
</tr>
<tr>
<td>message_types.FeaturesReplyBody</td>
<td>OK</td>
</tr>
<tr>
<td>message_types.EchoWithData</td>
<td>OK</td>
</tr>
<tr>
<td>message_types.GetConfigReply</td>
<td>OK</td>
</tr>
<tr>
<td>message_types.PacketInSizeMiss</td>
<td>OK</td>
</tr>
<tr>
<td>message_types.PacketInSizeAction</td>
<td>FAIL</td>
</tr>
<tr>
<td>message_types.PortModPacketIn</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.UdpDstPort</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.IpTos</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.WildcardMatchPrio</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.EthernetDstAddress</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.UdpSrcPort</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.ExactMatch</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.ICMPCode</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.MultipleHeaderFieldL2</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.MultipleHeaderFieldL4</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.VlanPCP</td>
<td>FAIL</td>
</tr>
<tr>
<td>flow_matches.EthernetSrcAddress</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.AllWildcardMatch</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.ICMPType</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.IngressPort</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.TcpSrcPort</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.TcpDstPort</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.ArpOpcode</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.ExactMatchPrio</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.ArpTargetIP</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.ArpSenderIP</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.VlanId</td>
<td>FAIL</td>
</tr>
<tr>
<td>flow_matches.IPProtocol</td>
<td>OK</td>
</tr>
<tr>
<td>flow_matches.EthernetType</td>
<td>OK</td>
</tr>
<tr>
<td>basic.EchoWithData</td>
<td>OK</td>
</tr>
<tr>
<td>basic.PacketInBroadcastCheck</td>
<td>OK</td>
</tr>
<tr>
<td>basic.DescStatsGet</td>
<td>OK</td>
</tr>
<tr>
<td>basic.PacketOutMC</td>
<td>OK</td>
</tr>
<tr>
<td>basic.PacketOut</td>
<td>OK</td>
</tr>
<tr>
<td>basic.PortConfigModErr</td>
<td>OK</td>
</tr>
<tr>
<td>basic.Echo</td>
<td>OK</td>
</tr>
<tr>
<td>basic.PortConfigMod</td>
<td>OK</td>
</tr>
<tr>
<td>basic.FlowMod</td>
<td>OK</td>
</tr>
<tr>
<td>basic.TableStatsGet</td>
<td>OK</td>
</tr>
<tr>
<td>basic.BadMessage</td>
<td>OK</td>
</tr>
</tbody>
</table>
Testing Intel® DPDK vSwitch with OFTest—Intel® DPDK vSwitch

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic.FlowStatsGet</td>
<td>OK</td>
</tr>
<tr>
<td>basic.PacketIn</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Delete_4</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Add_8</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Delete_1</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Delete_2</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Add_7</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Add_5</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Mod_3_Delete_1</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Mod_2</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Mod_3</td>
<td>OK</td>
</tr>
<tr>
<td>flow_query.Flow_Mod_1</td>
<td>OK</td>
</tr>
<tr>
<td>nicira_role.AnyReply</td>
<td>OK</td>
</tr>
<tr>
<td>flow_stats.TwoFlowStats</td>
<td>FAIL</td>
</tr>
<tr>
<td>flow_stats.AggregateStats</td>
<td>FAIL</td>
</tr>
<tr>
<td>flow_stats.EmptyFlowStats</td>
<td>FAIL</td>
</tr>
<tr>
<td>flow_stats.SingleFlowStats</td>
<td>FAIL</td>
</tr>
<tr>
<td>flow_stats.DeletedFlowStats</td>
<td>FAIL</td>
</tr>
<tr>
<td>flow_stats.EmptyAggregateStats</td>
<td>OK</td>
</tr>
<tr>
<td>counters.RxFrameErr</td>
<td>OK</td>
</tr>
<tr>
<td>counters.RxOErr</td>
<td>OK</td>
</tr>
<tr>
<td>counters.TxPktPerPort</td>
<td>FAIL</td>
</tr>
<tr>
<td>counters.PktPerFlow</td>
<td>OK</td>
</tr>
<tr>
<td>counters.RxCrcErr</td>
<td>OK</td>
</tr>
<tr>
<td>counters.TxBytPerQueue</td>
<td>OK</td>
</tr>
<tr>
<td>counters.RxErrors</td>
<td>OK</td>
</tr>
<tr>
<td>counters.BytPerFlow</td>
<td>OK</td>
</tr>
<tr>
<td>counters.RxBytPerPort</td>
<td>FAIL</td>
</tr>
<tr>
<td>counters.DurationPerFlow</td>
<td>OK</td>
</tr>
<tr>
<td>counters.LoupMatchedCount</td>
<td>FAIL</td>
</tr>
<tr>
<td>counters.ActiveCount</td>
<td>OK</td>
</tr>
<tr>
<td>counters.TxErrors</td>
<td>OK</td>
</tr>
<tr>
<td>counters.TxPktPerQueue</td>
<td>OK</td>
</tr>
<tr>
<td>counters.TxErrorPerQueue</td>
<td>OK</td>
</tr>
<tr>
<td>counters.RxPktPerPort</td>
<td>FAIL</td>
</tr>
<tr>
<td>counters.TxBytPerPort</td>
<td>FAIL</td>
</tr>
<tr>
<td>counters.RxDrops</td>
<td>OK</td>
</tr>
<tr>
<td>counters.Collisions</td>
<td>OK</td>
</tr>
<tr>
<td>counters.TxDrops</td>
<td>OK</td>
</tr>
</tbody>
</table>
12.0 Testing Intel® DPDK vSwitch with ovs-testsuite

The Open vSwitch testsuite is a set of unit tests that form the larger part of OVS' test framework. ovs-testsuite is an OVS-specific collection of unit tests that are designed to test atomic pieces of the code (as with all unit tests). This framework is written in GNU* Autotest,* with supporting Python* scripts and C unit test files.

12.1 Setup

The testsuite requires a minor amount of setup, due to the current design of Intel® DPDK vSwitch.

Many of the tests will not run "out of the box", as calls to many utilities used by the tests, such as ovs-ofctl, assume that said utilities are in the PATH. This is not the case as Intel® DPDK vSwitch does not "install" and place binaries in the /bin folder. As a result, the path must be modified, similar to the following:

```
export PATH=$PATH:$(OVS_DIR)/utilities:$(OVS_DIR)/ovsdb:$(OVS_DIR)/vswitchd
```

Before running tests, Intel® DPDK must be set up, as described in Section 5.1.1.3

12.2 Running Tests

Tests may be run using the Open vSwitch Makefile, or via the testsuite script generated by autoconf. Details of both methods follow.

12.2.1 Running Tests via Make

The full testsuite may be executed as follows:

```
    cd openvswitch
    make check
```

*Note:* As described in Section 2.7, test three (3) of the testsuite causes the switch to hang, and should be avoided.

*Note:* To speed up execution time, use the -j flag to run tests in parallel across all enabled cores. For example, to run all the Open vSwitch unit tests in a quad-core system (8 logical cores if Intel® HT Technology enabled):

```
    make check TESTSUITEFLAGS=-j8
```

To see a list of all the available tests, run:

```
    make check TESTSUITEFLAGS=--list
```

To run only a subset of tests, e.g. test 123 and tests 477 through 484:

```
    make check TESTSUITEFLAGS='123 477-484'
```
Tests do not have inter-dependencies, so any subset may be run independently.

To run tests matching a keyword, e.g. "ovsdb":

```bash
make check TESTSUITEFLAGS='--k ovsdb'
```

To see a complete list of test options:

```bash
make check TESTSUITEFLAGS=--help
```

### 12.3 Running Tests via the testsuite Script

It is also possible to run the tests via the testsuite script generated by autoconf. This is useful to run tests without rebuilding them (they are built automatically when Intel® DPDK vSwitch is built).

To run all the Open vSwitch unit tests in Intel® DPDK vSwitch, one at a time:

```bash
cd tests
./testsuite
```

To clean logs and results of previous tests:

```bash
./testsuite -c [tests]
```

**Note:** To speed up execution time, use the -j flag to run tests in parallel across all enabled cores. For example, to run all the Open vSwitch unit tests in a quad-core system (8 logical cores if Intel® HT Technology enabled):

```bash
./testsuite -j8
```

To see a list of all the available tests, run:

```bash
./testsuite --list
```

To run only a subset of tests, e.g., test 123 and tests 477 through 484:

```bash
./testsuite 123 477-484
```

To run tests matching a keyword, e.g., "ovsdb":

```bash
./testsuite -k ovsdb
```

To see a complete list of test options:

```bash
./testsuite --help
```
13.0 Intel® DPDK vSwitch Port Numbers

The various valid port values supported by Intel® DPDK vSwitch are summarized in the Table 6 below:

Table 6. Valid Port Value

<table>
<thead>
<tr>
<th>Port Type</th>
<th>Port Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>VirtIO</td>
<td>1–15</td>
</tr>
<tr>
<td>IVSHM</td>
<td>1–15</td>
</tr>
<tr>
<td>Physical</td>
<td>16-31</td>
</tr>
<tr>
<td>KNI</td>
<td>32-47</td>
</tr>
<tr>
<td>vEth</td>
<td>64-71</td>
</tr>
</tbody>
</table>