Intel® DPDK vSwitch

Getting Started Guide

Intel Corporation
Software Engineering

Copyright © 2013 Intel Corporation. All rights reserved

Intel is a trademark of Intel Corporation in the U.S. and/or other countries.

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

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 NEGLIGENT 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.

Revision History

Date	Revision	Description
March 2013	0.1.0	Initial version
April 2013	0.2.0	Updated with sections for IVSHM and 12-Tuple support
May 2013	0.3.0	Removed vswitchd instructions and changed ovs_dpdk parameters
June 2013	0.4.0	Updated to reflect directory naming changes and merge of QEMU
June 2013	0.5.0	Updated to add details of KNI configuration
June 2013	0.6.0	Updated to add Wind River Linux details and update method of mapping hugepages in the guest for IVSHM.
June 2013	0.7.0	Document restructuring and rework
July 2013	0.8.0	Updated to add changes for flow manipulation code
July 2013	0.9.0	Updates to flow manipulation sections and correction of errata
August 2013	0.9.1	Fixed broken links and added additional sample test setups

Contents

1 Introduction

This document contains detailed instructions for building and running the 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

These release notes cover modified Open vSwitch and QEMU packages that enable the use of Intel® Data Plane Development Kit (Intel® DPDK) to demonstrate performance and to be used as a reference architecture. This release adds support for dynamic flow modification, via the ovs-ofctl and ovs-dpctl commands.

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 out of the use of this software by anyone.

§ §

2 Release Notes & Known Bugs

2.1 Supported Operating Systems

This release has been validated against the following operating systems.

2.1.1 Host OS

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

2.1.2 Guest OS

- **Fedora 16** – Kernel 3.6.7-4

2.2 Supported Processors

This release has been validated on Intel® Sandy Bridge processors.

2.3 Intel® DPDK vSwitch

- This release supports DPDK v1.4.0 only.
- QEMU is added as a DPDK secondary process – attempting to run QEMU before ovs_dpdk will result in a segfault. This is standard DPDK behaviour.
- Intel Virtualization Technology for Directed I/O (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 could take place 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 is harmless, to suppress the warning the following command can be run:

  ```
  # 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-DPDK userspace applications in the guest), and offers significantly better performance compared to VirtIO. It is the preferred option when high throughput is required in a non-DPDK application use case.
- This release has not been tested or validated for this use with Virtual Functions although it should theoretically work with Intel DPDK v1.4.0.
- If testing performance with TCP, variances in performance may be observed – this is due to the protocol's congestion control mechanisms. UDP produces more reliable and repeatable results, and 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 a DPDK process starts, it attempts to reserve memory for various rings through a call to `rte_memzone_reserve`; in the case of a 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 a 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 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 lets the 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 DPDK `igb_uio` driver.
- The amount of time required to establish flows for exception packets (i.e. packets that don't match a switch flow table entry) has been observed to be of the order of 15 seconds in virtualized environments. This only affects cases in which flows have been added using `ovs-ofctl`.
- Sightings of issues including memory corruption and segfaults have been observed in the case of Virtio, but have not been reproduced to date.
- 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.

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.
- Changing the MTU or MAC address of a KNI device is currently unsupported.

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 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 > 256MB (1GB hugepage is used in IVSHM).

2.7 vswitchd

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

2.8 ovs-vsctl

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

2.9 ovs-ofctl

- Not all functionality that is supported by Open vSwitch is supported by the Intel DPDK vSwitch
- The only OpenFlow action currently supported is OUTPUT
- Matching on VLAN header fields (`dl_vlan`, `dl_vlan_pcp`) is not currently supported
- Flows must be added with ovs-ofctl while no traffic is traversing the switch. Attempting to add a flow while the switch is already processing traffic renders the daemon unresponsive to further `ovs-` commands.

2.10 ovs-dpctl

- Not all functionality that is supported by Open vSwitch is supported by the Intel DPDK vSwitch
- The only OpenFlow action currently supported is OUTPUT
- Matching on VLAN header fields (`dl_vlan`, `dl_vlan_pcp`) is not currently supported

3 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

Note: For required packages and instructions for Wind River Linux environment, please refer to to Section 6

Wind River Linux-Specific Setup.

3.2 Downloading Intel® DPDK

Download DPDK release package `DPDK.L.1.4.0-28.zip`:

- Existing customers can download the release package from the IBL 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 1.4.0 (L1.4.0-28) - Code (Zip File) and Intel Data Plane Development Kit (Intel DPDK) - Release 1.4.0 - Documentation
- Otherwise, please register on the Intel® Embedded website, and request design assistance for your project.

§ §

4 Compiling Intel® DPDK vSwitch from Source

4.1 Compilation of Intel® DPDK

Expand the Intel® DPDK release package:

```
mkdir ovs_dpdk
tar –C ovs_dpdk –xzvf <release_pkg_name>.tar.gz
```

Modify the DPDK buildsystem so that libraries are position independent:

```
--- 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
```

Compile Intel® DPDK. Please refer to the *DPDK Getting Started Guide* for details on how to do this.

4.2 Compilation of Open vSwitch

Once DPDK has been built, to build Open vSwitch perform the following steps:

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

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:

```
mkdir –p /usr/local/etc/openvswitch
mkdir –p /usr/local/var/run/openvswitch
```

4.3 Compilation of QEMU

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

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

§ §

5 Compiling and Running Sample Applications

5.1 VirtIO

This section contains instructions on how to compile and run a sample application which demonstrates performance of a 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 1GB, is used:

```
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* Compiling Intel® DPDK vSwitch from Source.

5.1.1.3 Setup DPDK

Once compilation of the above packages is complete, mount the 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, Configuring Intel® DPDK vSwitch Flow Tables using ovs-ofctl, for details on how to add ports to the switch, and section 11, Intel DPDK vSwitch Port Numbers, for details of the valid values for VirtIO ports.

5.1.1.5 Start Intel® DPDK vSwitch (ovs_dpdk)

Start the ovs_dpdk application

```
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 v1.4.0 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® DPDK 1.4.0 Getting Started Guide* 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 0x30 -n 4 --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 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:

```
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> -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>
```

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, Dynamic Flow manipulation using ovs-ofctl, and section 10, Dynamic Flow Manipulation using ovs-dpctl, for more information on programming flow tables, and section 5.6 Sample Test Setup (VM to VM via Virt-IO), for an example of how to program the switch's flow tables.

Add flows to switch traffic appropriately:

1. From ingress port to VirtIO port/ring used by VM (if using an external traffic generator)
2. From VirtIO port to next target (Physical/VirtIO/KNI/IVSHM port)
3. 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: In order to pass packets correctly between VMs, the flow table will need to 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 improves performance on the VM-VM path by ~10x over standard OVS using VirtIO.

This section contains instructions on how to compile and run a sample application which 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, Release Notes & Known Bugs 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 1GB, 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* Compiling Intel® DPDK vSwitch from Source.

5.2.1.3 Setup DPDK

Once compilation of the above packages is complete, insert the Intel DPDK kernel module and mount the hugepage – this will be mapped into 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 sections 9 and 10 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, Switch Manipulation using ovs-vsctl, for details on how to add ports to the switch, and section 11, Intel DPDK vSwitch Port Numbers, 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 v1.4.0 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® DPDK 1.4.0 Getting Started Guide* 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 towards 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, Dynamic Flow manipulation using ovs-ofctl, and section 10, Dynamic Flow Manipulation using ovs-dpctl, for more information on programming flow tables, and section 5.5 Sample Test Setup (Physical Port to VM to Physical port via IVSHMEM) for an example of how to program the switch's flow tables.

Add flows to switch traffic appropriately:

1. From ingress port to IVSHM port/ring used by VM
2. From IVSHM port to next target (Physical/Virtio/KNI/IVSHM port)
3. 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 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.

```
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 Huge Pages

Start the OS with the following added kernel options - this ensures that hugepages are enabled thus allowing the host's 1GB page to be used.

```
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 huge page 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:

```
ln -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 as shown in the next step.

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 Optimising Performance

To maximize throughput, individual cores should be assigned 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 HTT enabled)

5.2.3.1.1 Isolate cores (Instructions for Fedora 16)

Note: For the corresponding Wind River Linux steps, please refer to Section 6.6 Core Isolation, for more information.

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® HTT 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. Firstly, 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:

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

You can use the following table to guide you in affinitizing the host core:

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

Process	Core	Core Mask	Comments
Kernel	0	0x1	All other CPUs isolated (`isolcpus` boot parameter)
client_switching_core	1	0x2	Affinity set in `ovs_dpdk` command line
RX core	2	0x4	Affinity set in `ovs dpdk` command line
RX core	3	0x8	Affinity set in `ovs dpdk` command line
QEMU process VM1	4	0x10	`taskset -a <pid_of qemu_process>`
QEMU process VM1	5	0x20	`taskset -a <pid_of qemu_process>`
QEMU process VM2	6	0x40	`taskset -a <pid_of qemu_process>`
QEMU process VM2	7	0x80	`taskset -a <pid_of qemu_process>`

5.3 Intel DPDK KNI Setup

When created in a guest, KNI devices enable non-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 which, when run in the guest, allows the user to create an Intel DPDK KNI device, which will attach to queues in the host `ovs_dpdk` 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 v1.4.0.

5.3.1 Host Configuration

Follow the Host Configuration Guide given 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, etc.)

Additionally copy the KNI patch file to a temporary location:

```
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 Guest Configuration, 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 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 DPDK and inserting the KNI module.

```
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
```

5.3.2.2 Compile and run kni_client sample application

Copy the kni_client folder to a directory on the VM, compile, and run. 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 `ovs_dpdk` portmask. Refer to Section 7, *Intel® DPDK vSwitch Command Line Parameters*, 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)

In order to forward packets along the various datapaths supported by ovs_dpdk (physical port to physical port, physical port to VM, VM to physical port, VM to VM), the switch's flow table need to be programmed, either directly using ovs-dpctl, or indirectly using ovs-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 ovs-dpctl.

5.4.1 Test Setup

Initial setup, as described in the section of the relevant I/O method, and section 10, Dynamic Flow Manipulation using ovs-dpctl.

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 ./ovs-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),ipv
4(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 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.

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

```
sudo ./ovs-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),ipv
4(src=1.1.1.2,dst=1.1.1.1,proto=1,tos=0,ttl=64,frag=no)" "16"
```

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 back out to another physical port via the switch (Phy Port -> Switch -> VM -> Switch -> Phy Port), using ovs-dpctl.

5.5.1 Test Setup

Initial setup, as described in the IVSHM section, and section 10, Dynamic Flow Manipulation using ovs-dpctl.

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 dpdk@dp
"in_port(16),eth(src=00:00:00:00:00:11,dst=00:00:00:00:00:22),eth_type(0x0800),ipv
4(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.

In order 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 dpdk@dp
"in_port(1),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"
```

5.6 Sample Test Setup (VM to VM via Virt-IO)

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

Initial setup, as described in the Virtio section, and section 10, 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=outpu
t: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 2.2.2.1, 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 can also be useful for debugging.

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

In order 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=outpu
t:16
```

5.6.2.1 Configure VM Network Interfaces

Note: In order to pass packets correctly between VMs, the flow table will need to 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 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
```

6 Wind River Linux-Specific Setup

Compilation of the release in a Wind River environment differs slightly from standard distros. The relevant discrepancies are described in this section.

6.1 Layer

wr-intel-support

6.2 Template

feature/target-toolchain,feature/gdb,feature/intel-dpdk,feature/kvm

6.3 Required Packages

- qemu
- libvirt
- binutils-symlinks
- xz
- make
- glib-2.0-dev
- gtk+-dev
- glib-2.0-utils
- gdk-pixbuf-dev
- dbus-dev
- dbus-glib-dev
- alsa-dev
- curl-dev
- libxt-dev
- mesa-dri-dev
- rsync
- flex
- bison
- chkconfig
- patch
- git,vsftpd
- socat
- cmake,zlib
- automake
- autoconf
- libtool
- smartpm,
- openssh
- kernel-dev
- openssl-staticdev

Note: The `vlan` package will have to be imported after build to support VLAN. It can be found here: `http://www.candelatech.com/~greear/vlan/vlan.1.9.tar.gz`. Instructions on how to import the package can be found in the WRL install directory: `/opt/WRLinux_5.0.1/docs/extensions/eclipse/plugins/com.windriver.ide.doc.wr_linux_5/wr_linux_users_guide/index.html`

6.4 Compile Source

Before compiling the release source code, perform these steps:

```
export CC=gcc
ln -s /lib/modules/`uname -a`/build /usr/src/kernel
cd /usr/src/kernel
make scripts
```

6.5 Compile QEMU

An additional flag must be passed to 'configure' before compiling QEMU on Wind River Linux:

```
cd qemu
./configure --enable-kvm –target-list=x86_64-softmmu --dpdkdir=/path/to/dpdk
--extra-cflags="-Wno-poison-system-directories"
make
```

6.6 Core Isolation

To improve performance, isolate the majority of cores in the host from the kernel scheduler, and affinitize processes as previously described.

To perform CPU isolation in the host, edit `/boot/grub2/grub.cfg` (or `/etc/default/grub`, if applicable); specifically this line:

```
legacy_kernel
```

Include the following:

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

Note: `n` should be the max number of logical cores available (If Intel® HTT is enabled). Always leave core 0 for the operating system.

§ §

7 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 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, 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.

- `--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 Switch Manipulation using ovs-vsctl

8.1 ovs-vsctl

This release of Intel DPDK vSwitch adds support for 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, Configuring Intel® DPDK vSwitch Flow Tables using ovs-ofctl.

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 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 Dynamic Flow manipulation using ovs-ofctl

9.1 ovs-ofctl

This release of Intel DPDK vSwitch adds support for 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:

```
ovs-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 not supported in this release

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

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 action supported is `output`. For example, where outputting 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
```

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:

Command	Comments
`in_port=port`	Datapath in port
`dl_src=XX:XX:XX:XX:XX:XX`	Source MAC
`dl_dst=XX:XX:XX:XX:XX:XX`	Destination MAC
`dl_type=ethertype`	Ethernet protocol type
`nw_src=ip`	Source IP
`nw_dst=ip`	Destination IP
`nw_proto=proto`	IP protocol type
`nw_tos=tos`	IP ToS
`tp_src=port`	UDP or TCP source port
`tp_dst=port`	UDP or TCP destination port

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

In order to use `ofctl`, the `vswitch` daemon needs to 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. 16-31 refer to physical 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

```
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 -k 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. In order to prevent this you must first delete this flow table entry.

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

Then configure 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. it only implements the Openflow `OUTPUT` action. Ports 1-15 refer to VirtIO/IVSHM rings. 16-31 refer to physical ports. 32-47 refer to KNI FIFOs.

- `<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`, `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=out
put:16
```

§ §

10 Dynamic Flow Manipulation using ovs-dpctl

10.1 ovs-dpctl

This release of Intel DPDK vSwitch adds support for 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.

ovs-dpctl syntax is generally in the format:

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

Where:

- `DPDK_OPTION`: Specify options specific to the DPDK library. Currently:
  - `-c <core_mask>`
  - `-proc-type=secondary`
- `OPTIONS`: One of the options specified in Section 10.3
- `COMMAND`: One of the commands specified in Section 10.2.
- `ARG`: One of the additional arguments specified in Section 10.2 Supported Commands

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: Matching on VLAN header fields (`vlan(vid=vlan_id,pcp=vlan_prio)`) is not supported in this release

10.2 Supported Commands

10.2.1 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,frag=no)" "17"
```

10.2.2 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,frag=no)" "17"
```

10.2.3 del-flows

Delete all flows from DP

```
del-flows DP
```

Example:

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

10.2.4 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
```

10.2.5 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"
```

10.2.6 dump-flows

Display flows in DP

```
dump-flows DP
```

Example:

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

10.3 Supported Options

The following flow strings are supported:

Command	Comments
`-s, --statistics`	Print statistics for flow • For commands 'dump-flows' and 'get-flow', stats are always printed, even without this option • For 'add-flow' prints zero stats • For 'mod-flow' prints stats before modification • For 'del-flow' prints stats before deletion • No action for del-flows
`--may-create`	Create flow if it doesn't exist*
`--clear`	Reset existing stats to zero*

* used only with mod-flow command

10.4 Supported Arguments

The following arguments are supported:

Command	Comments
`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	The output port number

*The following flow modifier strings are supported

- in_port(port_id)
- eth(src="XX;XX:XX:XX:XX:XX",dst="YY:YY:YY:YY:YY:YY")
- eth_type(ethertype)
- 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)

10.5 Configuring Intel® DPDK vSwitch Flow Tables Using ovs-dpctl

ovs-dpctl adds exact match flow table entries directly into 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 can be useful for debugging. See the ovs-dpctl man page for more information.

To use ovs-dpctl, ensure that the ovs_dpdk application is running, and that the vswitchd is not running.

Flows can be added using the following command:

```
sudo ./utilities/ovs-dpctl -c 1 --proc-type=secondary -- -s add-flow dpdk@dp
"<FLOW>" "<ACTION>"
```

Flows can be deleted using the following command:

```
sudo ./utilities/ovs-dpctl -c 1 --proc-type=secondary -- -s del-flow dpdk@dp
"<FLOW>"
```

Flows can be modified using the following command (clear will optionally clear the statistics):

```
sudo ./utilities/ovs-dpctl -c 1 --proc-type=secondary -- -s mod-flow dpdk@dp
"<FLOW>" "<NEW_ACTION>" -clear
```

All flows can be dumped using the following command:

```
sudo ./utilities/ovs-dpctl -c 1 --proc-type=secondary -- -s dump-flows
dpdk@dp
```

One flow can be retrieved using the following command:

```
sudo ./utilities/ovs-dpctl -c 1 --proc-type=secondary -- -s get-flow dpdk@dp
"<FLOW>"
```

All flows can be deleted using the following command

```
sudo ./utilities/ovs-dpctl -c 1 --proc-type=secondary -- -s del-flows
dpdk@dp
```

An example of adding a flow table entry can be seen here:

```
# ./utilities/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=1,tos=0,ttl
=64,frag=no)" "17"
```

Note: The –s parameter toggles statistics printing for each ovs-dpctl operation

Note: As previously mentioned, if a flow field is not supplied to ovs-dpctl, it is assumed to be zero - in the command above, the destination and source MAC address are not supplied, and as such are assumed to be 00:00:00:00:00:00

11 Intel DPDK vSwitch Port Numbers

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

I/O Method	Port Range
VirtIO	1 – 15
IVSHM	1 – 15
Physical Ports	16-31
KNI Devices	32-47