Intel® DPDK vSwitch

Getting Started Guide

Intel Corporation
Software Engineering

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

Table of Contents

Revision History 2

Definitions and Acronyms 5

1. Introduction 5

2. Description of Release 5

3. Release Notes & Known Bugs 6

3.1 Supported Operating Systems 6

3.2 Supported Processors 6

3.3 Intel® DPDK vSwitch 6

3.4 Intel® DPDK vSwitch Sample Guest Applications 6

3.5 Open vSwitch 6

3.6 QEMU 7

4. Setup 8

4.1 Required Libraries 8

4.2 Initial Setup 8

4.3 Virtio Setup 9

4.3.1 Compile Intel® DPDK 9

4.3.2 Compile Open vSwitch 9

4.3.3 Compile QEMU 9

4.3.4 Start Intel® DPDK vSwitch(ovs_dpdk) 9

4.3.5 Start QEMU 10

4.3.6 Configure VM Network Interfaces 10

4.4 IVSHM Setup 11

4.4.1 Host Configuration 11

4.4.2 Guest Configuration 13

4.4.3 Optimizing Performance 14

4.5 Intel® DPDK KNI Setup 16

4.5.1 Host Configuration 16

4.5.2 Guest Configuration 16

5. Wind River Linux-Specific Setup 17

5.1 Layer 17

5.2 Template 17

5.3 Required Packages 17

Compile Source 18
5.4 18
5.4.1 Compile QEMU 18
5.5 Core Isolation 18
6. Intel ® DPDK vSwitch (ovs_dpdk) Command Line Parameters 19
7. Intel ® DPDK vSwitch (ovs_dpdk) Flow Tables 20

Definitions and Acronyms

Acronym	*Description*
DPDK	*Intel® Data Plane Development Kit*
OVS	*Open vSwitch*
OVS DPDK	*DPDK Sample Application that supports OVS integration*
VM	*Virtual Machine*
IVSHM	*Inter-Virtual-Machine Shared Memory*
Host	*Physical machine/Operating System, on which Virtual Machines are run*
Guest/Client	*Virtual Machine/Operating System which runs on top of a Host system*
HTT	*Intel® Hyper-Threading Technology*
KNI	*Kernel NIC Interface*

1. Introduction

This document contains detailed instructions for building and running the OVS DPDK software. It describes how to compile and run OVS DPDK, QEMU and guest applications in a Linux environment.

2. 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. The release supports numerous IO virtualization mechanisms: Virtio, IVSHM, and IVSHM with Intel® DPDK KNI.

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.

3. Release Notes & Known Bugs

3.1 Supported Operating Systems

- This release has been validated against the following Operating Systems.

Host OS

- Fedora 16 (kernel 3.6.7-4)
- Wind River Linux 5.0.1.0_standard (kernel 3.4.34)

Guest OS

- Fedora 16 (kernel 3.6.7-4)

3.2 Supported Processors

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

3.3 Intel® DPDK vSwitch

- Runtime modification of the flow tables is currently unsupported. Details of the static flow table entries included in the switch are listed in section 7, Intel® DPDK vSwitch (ovs_dpdk) Flow Tables.
- The only traffic types supported are TCP/IP and UDP/IP. ARP is not supported, so any traffic generator used to pass packets for PHY-PHY tests must not send ARP packets first.
- Packets must be sent in multiples of 32 to see zero packet loss
- 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.
- VT-d must be disabled in the BIOS when using DPDK versions prior to 1.4. Versions of DPDK prior to 1.4 have an issue that can cause problems using a PMD with a NIC PF when VT-d is enabled.
- 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 should be harmless, to suppress the warning the following command can be run:

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

3.4 Intel® DPDK vSwitch Sample Guest Applications

- 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.
- The IP mangling in the IVSHM ovs_client presumes an offset for Ethernet header only. Adding a VLAN tag to this traffic will cause it to fail.
- Changing the MTU or MAC address of a KNI device is currently unsupported.
- The KNI guest application is not supported by Intel® DPDK v1.3.1.
- Multiple KNI devices in a single guest use case has not been validated.

3.5 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
- The ofctl and vsctl utilities are currently unsupported

3.6 QEMU

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

4. Setup

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

Note: VT-d must be disabled in the BIOS when using Intel® DPDK versions prior to 1.4. Versions of DPDK prior to 1.4 have an issue that can cause problems using a PMD with a NIC PF when VT-d is enabled.

4.1 Required Libraries

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

1. gcc
2. kernel-dev
3. kernel-devel
4. coreutils
5. make
6. nasm
7. glibc-devel.i686
8. libc6-dev-i386
9. glibc-devel.x64_86
10. glibc-devel
11. kernel-devel-3.3.4-5.fc17.x86_64 (matching kernel)
12. autoconf
13. automate
14. autom4te
15. automake
16. zlib-devel
17. glib2-devel.x86_64

For required packages for Wind River Linux, please refer to section 5, Wind River Linux-Specific Setup

4.2 Initial Setup

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.

Note: This release has also been minimally tested, but not validated, against Intel® DPDK release package IntelDPDK.L.1.3.1_7.zip, available here. Please note that this version of Intel® DPDK does not support all of the features contained within this release – refer to section 3, Release Notes & Known Bugs for details.

4.3 Virtio Setup

This section describes how to compile and run the Intel® DPDK vSwitch and its related applications, using a DPDK-accelereated version of Virtio for IO virtualization.

4.3.1 Compile 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
```

Please refer to the DPDK Getting Started Guide for details on how to compile Intel® DPDK.

4.3.2 Compile Open vSwitch

```
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

4.3.3 Compile QEMU

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

4.3.4 Start Intel® DPDK vSwitch(ovs_dpdk)

```
sudo ./openvswitch/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: Please refer to section 6, Intel® DPDK vSwitch (ovs_dpdk) Command Line Parameters for details on command-line parameters

4.3.5 Start QEMU

Note: QEMU will fail if ovs_dpdk is not already running

Start VM1:-
qemu/x86_64-softmmu/qemu-system-x86_64 -c 0x30 -n 4 --proc-type=secondary -- -boot c -hda <PATH_TO_IMAGE>.qcow2 -m 512M -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" -mem-path /mnt/huge -mem-prealloc

Start VM2:-
qemu/x86_64-softmmu/qemu-system-x86_64 -c 0xC0 -n 4 --proc-type=secondary -- -boot c –hda <PATH_TO_IMAGE>.qcow2 -m 512M -netdev dpdk,port=2,id=me2 -device virtio-net-pci,netdev=me2,mac=00:00:00:00:00:02 -smp 2 -enable-kvm -name "Client 2" -mem-path /mnt/huge -mem-prealloc

4.3.6 Configure VM Network Interfaces

Note: the VM's must have the 'vlan' package installed

In VM1's terminal:
modprobe 8021q
ifconfig eth0 up
vconfig add eth0 700
ifconfig eth0.700 2.2.2.1/24 up
arp -s 2.2.2.2 00:00:00:00:00:02

In VM2's terminal:
modprobe 8021q
ifconfig eth0 up
vconfig add eth0 700
ifconfig eth0.700 2.2.2.2/24 up
arp -s 2.2.2.1 00:00:00:00:00:01

4.4 IVSHM Setup

Intel® DPDK vSwitch supports the mapping of a host-created Intel® DPDK hugepage directly into guest userspace, thus 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 utilize 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 3, Release Notes & Known Bugs for details.

4.4.1 Host Configuration

Configure Kernel Boot Parameters

- Start the O/S with the following added kernel boot options – this ensures that a single hugepage, of size 1GB, is used:

```
default_hugepagesz=1G hugepagesz=1G hugepages=1
```

Setup DPDK

- Compile Intel® DPDK, as specified in section 4.3.1.
- 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.

Build Source Code

- Compile Open vSwitch and QEMU, as specified in section 4.3, Virtio Setup.

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: Intel® DPDK v1.3.0 and below automatically unbinds drivers for any supported NIC ports found – to avoid this issue, blacklist ports if necessary, using the –b flag:

```
sudo ./openvswitch/datapath/dpdk/build/ovs_dpdk -c <core_mask> -n 4 [-b <list of black-listed port PCI ID’s>] --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 -b 0000:0b:00.0 -b 0000:0b:00.1 -b 0000:0b:00.2 -b 0000:0b:00.3 --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 2 VMs, a value of 3 should be used as the ‘number_of_clients’ parameter.

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
```

Start QEMU

- Start QEMU on the host

```
./qemu/x86_64-softmmu/qemu-system-x86_64 –c <coremask> -n 4 -- –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=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 –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=fat:/tmp/share,snapshot=off
```

4.4.2 Guest Configuration

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

Enable Huge Pages

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

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

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
```

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
```

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
```

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
```

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>
```

Note: Client ID 0 is reserved for the vswitchd interface, and should not be used

4.4.3 Optimizing Performance

In order 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.

Sample setup for 8-core system (16 logical cores if Intel HTT enabled)

Isolate cores (Instructions for Fedora 16)
Note: For the corresponding Wind River Linux steps, please refer to section 5, Wind River Linux-Specific Setup.

1) In the host, edit /boot/grub2/grub.cfg (or /etc/default/grub, if applicable), appending the line marked

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

to include

```
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.

2) Update the grub configuration file

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

3) Reboot the system

ovs_client Commands

```
ovs_client -c 0x1 -n 4 --proc-type=secondary -- -n 1 -s -x02020201 -d 0x01010101
ovs_client -c 0x1 -n 4 --proc-type=secondary -- -n 2 -s -x02020202 -d 0x01010102
```

The -s and -d switches describe the hexadecimal value of the source and destination IP addresses that the ovs_client updates incoming packets with, before returning them to the switch.

ovs_dpdk 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)"
```

Core Affinity

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	Affinity set with taskset –a <pid_of_qemu_ process>
QEMU process VM1	5	0x20	Affinity set with taskset –a <pid_of_qeum_process>
QEMU process VM2	6	0x40	Affinitity set with taskset -a <pid_of qemu_process>
QEMU process VM2	7	0x80	Affinitity set with taskset -a <pid_of qemu_process>

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

4.5 Intel® DPDK KNI Setup

When created in a guest, KNI devices enable non-DPDK applications running in the VM to utilize the Intel® DPDK shared hugepage via 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 utilize KNI.

Note: This release only supports the KNI implementation contained within Intel® DPDK 1.4.

4.5.1 Host Configuration

Follow the "Host Configuration" steps as described in section 4.4.1.

Additionally, copy the KNI patch file to the temporary location:

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

4.5.2 Guest Configuration

Follow the "Guest Configuration" steps as described in section 4.4.2, up until the "Compile and run ovs_client sample application" step.

Insert the rte_kni module

A small number of modifications to the standard DPDK driver are required to support KNI devices in the guest that use the Intel® DPDK hugepage via the IVSHM model – these changes have been included as a patch. Apply the kni_misc path, 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
```

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 one of the KNI devices configured in the
portmask
```

Note: kni portmask above is similar to the ovs_dpdk portmask - refer to section 6, Intel® DPDK vSwitch (ovs_dpdk) Command Line Parameters, for details. However, the kni_portmask should not be entered in **decimal format only** (i.e. no prepending '0x').

5. 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.

5.1 Layer

wr-intel-support

5.2 Template

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

5.3 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

Note The vlan package will have to be imported after build in order 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

5.4 Compile Source

Before compiling the release source code, perform the following steps:

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

5.4.1 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
```

5.5 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), appending the line marked

```
legacy_kernel
```

to include

```
'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.

6. Intel ® DPDK vSwitch (ovs_dpdk) 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

  The number of supported clients

- -p

  Portmask - a hexadecimal bitmask representing the ports to be configured

  Each bit in the mask represents a port ID (bit 0=port0, bit 1=port1, etc.)
  e.g. by specifying a portmask of 0x3, ports 0 and 1 are configured

- -k

  KNI portmask - 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.

- --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.

7. Intel ® DPDK vSwitch (ovs_dpdk) Flow Tables

The current implementation of Intel® DPDK vSwitch does not support dynamic addition/deletion of flow table entries. The static entries contained within the switch's flow tables are summarized below.

Ingress Port	Source IP Address	Destination IP Address	Egress Port
DPDK Port 0	1.1.1.1	1.1.1.2	DPDK Port 1
DPDK Port 0	1.1.1.1	2.2.2.1	Client 1
DPDK Port 0	1.1.1.1	2.2.2.2	Client 2
DPDK Port 0	1.1.1.1	3.3.3.1	KNI Port 0
DPDK Port 0	1.1.1.1	3.3.3.2	KNI Port 1
DPDK Port 1	1.1.1.2	1.1.1.1	DPDK Port 0
DPDK Port 1	1.1.1.2	2.2.2.1	Client 1
DPDK Port 1	1.1.1.2	2.2.2.2	Client 2
DPDK Port 1	1.1.1.2	3.3.3.1	KNI Port 0
DPDK Port 1	1.1.1.2	3.3.3.2	KNI Port 1
Client 1	2.2.2.1	1.1.1.1	DPDK Port 0
Client 1	2.2.2.1	1.1.1.2	DPDK Port 1
Client 1	2.2.2.1	2.2.2.2	Client 2
Client 1	2.2.2.1	3.3.3.1	KNI Port 0
Client 1	2.2.2.1	3.3.3.2	KNI Port 1
Client 2	2.2.2.2	1.1.1.1	DPDK Port 0
Client 2	2.2.2.2	1.1.1.2	DPDK Port 1
Client 2	2.2.2.2	2.2.2.1	Client 1
Client 2	2.2.2.2	3.3.3.1	KNI Port 0
Client 2	2.2.2.2	3.3.3.2	KNI Port 1
KNI Port 0	3.3.3.1	1.1.1.1	DPDK Port 0
KNI Port 0	3.3.3.1	1.1.1.2	DPDK Port 1
KNI Port 0	3.3.3.1	2.2.2.1	Client 1
KNI Port 0	3.3.3.1	2.2.2.2	Client 2
KNI Port 0	3.3.3.1	3.3.3.2	KNI Port 1
KNI Port 1	3.3.3.2	1.1.1.1	DPDK Port 0
KNI Port 1	3.3.3.2	1.1.1.2	DPDK Port 1
KNI Port 1	3.3.3.2	2.2.2.1	Client 1
KNI Port 1	3.3.3.2	2.2.2.2	Client 2
KNI Port 1	3.3.3.2	3.3.3.1	KNI Port 0