# DPDK Functional Verification Instructions

| | |
|---|---|
| Software version | 16.11 |
| Document revision | 2.1 |

## Table of Contents

## Document Revision History

| Revision | Date | Description |
|---|---|---|
| 2.1 | 28 Feb 2017 | Document update for Intel XL710 (i40e) in Ubuntu 17.04 |
| 2.0 | 05 Oct 2016 | Document update for Ubuntu 16.10 |
| 1.0 | 24 Jul 2016 | Additional test instructions |
| 0.9 | 18 May 2016 | Initial revision |

## Document Contributions

| Revision | Author | Description |
|---|---|---|
| 2.1 | Gowrishankar Muthukrishnan *gomuthuk@in.ibm.com* | Document update |
| 2.0 | Gowrishankar Muthukrishnan *gomuthuk@in.ibm.com* | Document update |
| 1.0 | Gowrishankar Muthukrishnan *gomuthuk@in.ibm.com* | Document update |
| 0.9 | Gowrishankar Muthukrishnan *gomuthuk@in.ibm.com* | Initial revision |
| | Pradeep Satyanarayana *pradeep@us.ibm.com* | |

# 1. Scope

Purpose of this document is to provide a brief introduction and guidance to Functional verification testing (FVT) team and other test teams about Data Plane Development Kit (DPDK) which is shipped along with Ubuntu distributions. DPDK software version is listed on top of this document. Testing instructions are prepared within product capabilities of said software version.

## Introduction

Bridges in networking consist of a control plane and a data plane. Control plane is involved in the routing and traffic engineering aspects. Data plane simply forwards the packets. In traditional switches like a Linux bridge, there is no separation of the control plane and the data plane.

Software Defined Networking (SDN) and newer cloud workloads have contributed to the separation of the control plane and the data plane. Open vSwitch (OVS) is an example of an implementation that separates the control and data planes.

However, OVS design requires crossing the user-kernel boundaries multiple times and this contributes to the additional latencies. DPDK is designed to accelerate OVS and reduce latencies.

DPDK accomplishes this by pre-allocating the buffers, reducing locking overhead and using huge pages thereby reducing the address translations required. All of these help with lowering the latency. Polling is also used to reduce the interrupt load on the system. These are some of the techniques used to accelerate OVS.

# 2. Software setup and configurations

Backbones of DPDK packet forwarding acceleration are dedicated CPUs for DPDK, poll mode driver (PMD) threads, huge pages reservation in system memory and necessary virtio networking configurations (in case virtualization with KVM is considered).

Minimum number of hugepages (of size 16MB) needed to start a DPDK PMD thread is listed as below.

| Network Adapter | Minimum Hugepages Needed |
|---|---|
| Mellanox Connect X4 | 3 |
| Intel XL710 | 64 |

Although there is virtually no limit on its upper boundary for hugepage reservation (without exceeding physical memory), your system configuration (NUMA) and its hugepages allocation (i.e boot time / post boot) play critical roles on capping hugepage reservation for DPDK, to successfully allocate necessary memory pool, lockless rings and packet buffers for packets processing. At the maximum, we advice to reserve upto 8192MB to stick with guaranteed hugepages allocation by a NUMA system when there is no local memory for at least one CPU socket. You may use sysctl to allocate hugepage memory after booting as well, but it is strongly advised to use kernel parameters to guarantee contiguous allocations during boot time.

```
# open /etc/default/grub and edit GRUB_CMDLINE_LINUX to append
hugepages=512

# update grub.cfg to reflect above new parameter
update-grub
```

Below are the steps in case to reserve hugepage memory using sysctl (after system boots).

```
# reserve 8192MB hugepage memory using 16MB pages.
echo 512 > /sys/kernel/mm/hugepages/hugepages-16384kB/nr_hugepages

# In case of NUMA system, per numa node (N1 and N2) allocation can
# also be achieved as below.
echo 256 > /sys/devices/system/node/node<N1>/hugepages/hugepages-
           16384kB/nr_hugepages
echo 256 > /sys/devices/system/node/node<N2>/hugepages/hugepages-
           16384kB/nr_hugepages
```

Once system booted, update /etc/dpdk/dpdk.conf for NR_16M_PAGES to be 512 and start dpdk service.

```
service dpdk start
service dpdk status
```

Of all the guest operating systems that Ubuntu supports, we keep guest OS of same version as in host, for documentation and demonstration of virtualization scenarios. Similar (or with some additional vendor specific) instructions may be required to setup any other supported guest OS inside VM. If there are any DPDK specific instructions for other guest OS, we mention them in concerned use cases in this document, or otherwise refer to troubleshooting section for possible common issues.

This revision of document does not cover usecases of running DPDK inside any supported guest OS. But any guest OS would eventually need to have hugepage memory backend for working with DPDK vhost libraries. Hence, sufficient hugepages need to be reserved in host for both DPDK and qemu VMs.

### 2.1 Software repository information

For Ubuntu 16.10 in ppc64le, DPDK is available as PPA package. Follow below steps to install dpdk and openvswitch-switch-dpdk:

```
add-apt-repository ppa:paelzer/deb-dpdk-16.07
apt-get update
apt-get install dpdk* librte-pmd-* openvswitch-switch-dpdk

update-alternatives --set ovs-vswitchd /usr/lib/openvswitch-switch-
dpdk/ovs-vswitchd-dpdk
```

For Ubuntu 17.04 in ppc64le, DPDK is available as {edit}main{edit} package. Follow below steps to install dpdk and openvswitch-switch-dpdk:

```
apt-get update
apt-get install dpdk* librte-pmd-* openvswitch-switch-dpdk

update-alternatives --set ovs-vswitchd /usr/lib/openvswitch-switch-
dpdk/ovs-vswitchd-dpdk
```

### 2.2 Binding kernel modules

Some of the poll mode drivers including i40e (Intel XL710) needs vfio-pci or igb_uio kernel module loaded on network adapter to facilitate PCI probing in userspace (UIO).

```
# Intel XL710 (i40e) PMD
sudo dpdk-devbind –bind=vfio-pci <PCI ID of ports>

# To roll back from vfio-pci
sudo dpdk-devbind –bind=i40e <PCI ID of port 1>
```

# 3. DPDK FVT tests

DPDK can be run as stand alone PMD threads for Virtual machines or Applications to process packets at very high speed over network interface card (NIC). DPDK PMD threads can also be run with OVS vswitch daemon (ovs-vswitchd) to offload fast path from kernel into PMD thereby, OVS ports can communicate each other at high speed. Hence, we categorize FVT tests into standalone DPDK and OVS DPDK tests.

## 3.1 Standalone DPDK tests

Objective of these tests are to test DPDK for below use cases:

1. Check PMD cpu assignments
2. Check PMD socket memory allocations
3. Check PMD on SRIOV VF for packet forwarding
4. Check two VMs communicating through vhost PMD
5. Check two VMs communicating through SRIOV PMD
6. Check ip_pipeline sample application for LPM, ACL vector libraries

### 3.1.1 Check PMD cpu assignments

DPDK application can be instructed to start Poll Mode Driver threads on  specific CPU logical cores (lcores). Hence, its environment abstraction library (EAL) parameter “-c” can be used to pass CPU core mask (in hexa decimal). There is also another similar parameter “-l” to pass CPU core mask in numbers. At a minimum, you would need to pass two lcores – one act as master lcore and rest as forwarding lcores.

For eg, in below test we request DPDK to start PMD threads on CPU lcore 0,8 and 16. In this, lcore 0 is used as master lcore and other two are forwarding lcores. All DPDK library initialization (memory pool creation, packet buffer allocations, PCI device probe and initializing application) are handled within master lcore and then all lcores participate in application processing.

In order for testpmd sample application to probe network adapter ports, we need to also pass “-d” parameter pointing to its poll mode driver as in below table.

| Network Adapter | Poll Mode Driver (PMD) |
|---|---|
| Mellanox Connect X4 | librte_mlx5_pmd.so |
| Intel XL710 | librte_i40e_pmd.so |

In Ubuntu OS, its installation location is “/usr/lib/powerpc64le-linux-gnu/” and hence, not repeated mentioning everywhere in following sections.

```
# use CPU core mask
testpmd -d <librte_X_pmd.so> -c 0x10101

# use CPU core number
testpmd -d <librte_X_pmd.so> -l 0,8,16
```

As PMD starts, in same console (where you run testpmd), through “show config” command you would notice below portion of log that confirms DPDK threads are running on intended forwarding lcores.

```
testpmd> show config cores
List of forwarding lcores:  8 16
testpmd>
```

You may also use EAL parameter “--log-level 8” for verbose logging of EAL and PMD while initializing DPDK.

Refer to DPDK issues in troubleshooting section for common issues related to CPU lcore assignments.

### 3.1.2 Check PMD socket memory allocations

As part of bringing up DPDK PMD threads, EAL initiates memory pool creation for packet buffers. Unless specific parameters (as below) mentioned to EAL, DPDK attempts to map all hugepages available within system into its process space and give them to other libraries relying on it.

```
# DPDK app without memory parameters
testpmd -d <librte_X_pmd.so> -c 0x10101

# Notice below log in output
USER1: create a new mbuf pool <mbuf_pool_socket_0>: n=163456,
size=2176, socket=0
```

In above, 163456 mbufs of size 2176 bytes are created considering 32 ports at the maximum available in system and 2048 descriptors per rx/tx path of each port and 250 mbufs per forwarding lcore for each port and finally 512 mbufs for max packet burst : (2048+2048+(250 * 2)+512) * 32.

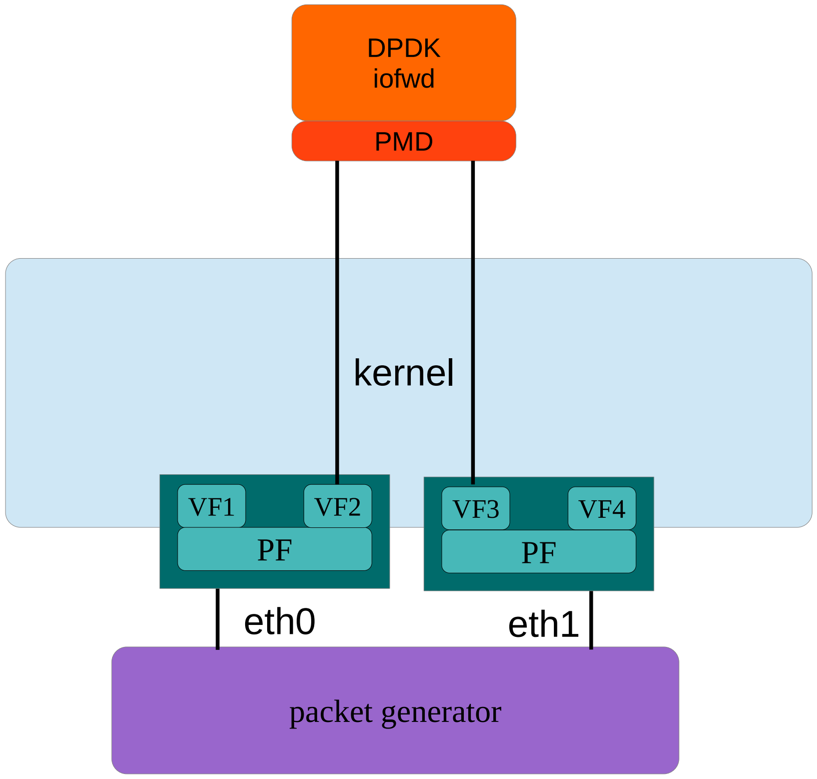
You may use EAL parameter “--socket-mem” to control DPDK to reserve only specific memory (in MB) from hugetlbfs. Similarly, testpmd app provides its own parameter “--total-num-mbufs” (in number) to control memory that each port needs within this memory pool, for handling transmission and reception of packets. In this version of application, total-num-mbufs to be more than 1024 (Mellanox CX4) or 1360 (Intel XL710) and a socket-mem to be 16MB as minimum as DPDK also needs some portions of socket pool memory for storing its logging and device data.

```
# DPDK app with memory parameters
testpmd -d <librte_X_pmd.so> -c 0x10101 ––socket-mem=16 -- ––total-
num-mbufs=1025

# Notice below log in output
USER1: create a new mbuf pool <mbuf_pool_socket_0>: n=1025, size=2176,
socket=0
```

Refer to DPDK issues in troubleshooting section for common issues related to hugepage reservations.

### 3.1.3 Check PMD on SRIOV VF for packet forwarding

This test verifies whether DPDK could start poll mode driver on SRIOV Virtual Functions (VF) and there by, acting as packet forwarding application between two Physical functions (PF). In the figure below, eth0 and eth1 mean port 0 and port 1 of network adapter for the demonstration of this test.



Setup instructions are given below. It is assumed that, you have followed SRIOV configuration guide as per network adapter you test.

```
# Mellanox CX4 SRIOV instructions
https://community.mellanox.com/docs/DOC-2386
https://community.mellanox.com/docs/DOC-2473

# Intel XL710 SRIOV instructions
http://www.intel.com/content/dam/www/public/us/en/documents/technology
-briefs/xl710-sr-iov-config-guide-gbe-linux-brief.pdf
```

Also, ensure VF2 and VF3 are enabled for unmatched traffic to pass through. VF1 and VF4 can be left untouched (Instructions as below respective adapters).

```
# Mellanox CX4:
# Enable privileged mode (on VF2 of port 0 and VF3 pf port 1)
ip link set <port 0 name> vf 1 trust on
ip link set <port 1 name> vf 0 trust on

# Accept unmatched traffic
ifconfig <VF2 interface name> promisc
ifconfig <VF3 interface name> promisc
```

```
# Intel XL710:
# Disable spoofcheck (on VF2 of port 0 and VF3 pf port 1)
ip link set <port 0 name> vf 1 spoofchk off
ip link set <port 1 name> vf 0 spoofchk off
```

In below setup instructions. testpmd uses EAL parameter "-w" which is to choose (i.e, whitelist) PCI devices for DPDK application to further configure PMD threads. As we need two ports for packet forwarding, we chose port mask 0x3 i.e out of detected ports 0,1,2,3,.. use port 0 and 1 for packet forwarding. Port mask is similar to CPU core mask as explained in Section 3.1.1.

```
# Start dpdk forward app on VF2 and VF3
testpmd -c 0x10101  \
      -d <librte_X_pmd.so> \
      -w <VF2 PCI Addr> -w <VF3 PCI Addr>
      -- -i –portmask=0x3

# Enter "start" in testpmd interactive shell.
```

To verify functioning of VF PMD, we send few streams of TCP traffic generated through iperf tool from a remote system (having connected two physical ports in a loop back) and see if iperf completes its tests.

Before starting tests in the remote system, create a network namespace and hide its second physical port inside it. This is to avoid IP route lookup within same system as both port IP addresses are within same node.

```
# configure port 1
ip addr add 10.4.4.3/24 dev enP2p1s0f0
ip link set dev enP2p1s0f0 up

# configure port 2
ip netns add tcp-server
ip link set enP2p1s0f1 netns tcp-server
ip netns exec tcp-server ip addr add 10.4.4.4/24 dev enP2p1s0f1
ip netns exec tcp-server ip link set dev enP2p1s0f1 up
```

Run iperf tests between two ports in remote system.

```
# Create tcp traffic on one port
ip netns exec tcp-server iperf -s

# Receive tcp traffic on other port
iperf -c 10.4.4.4
```

In above tests, verify whether DPDK dropped any packets. You can run below stats command in PMD shell (as mentioned in the beginning of this section).

```
# Report packets stat port 0. Use “all” instead of 0 for all ports

testpmd> show port stats 0

  ######################## NIC statistics for port 0
########################
  RX-packets: 306399     RX-missed: 0          RX-bytes:  18847534218
  RX-errors: 0
  RX-nombuf:  0
  TX-packets: 57930      TX-errors: 0          TX-bytes:  3823468

######################################################################
######
```

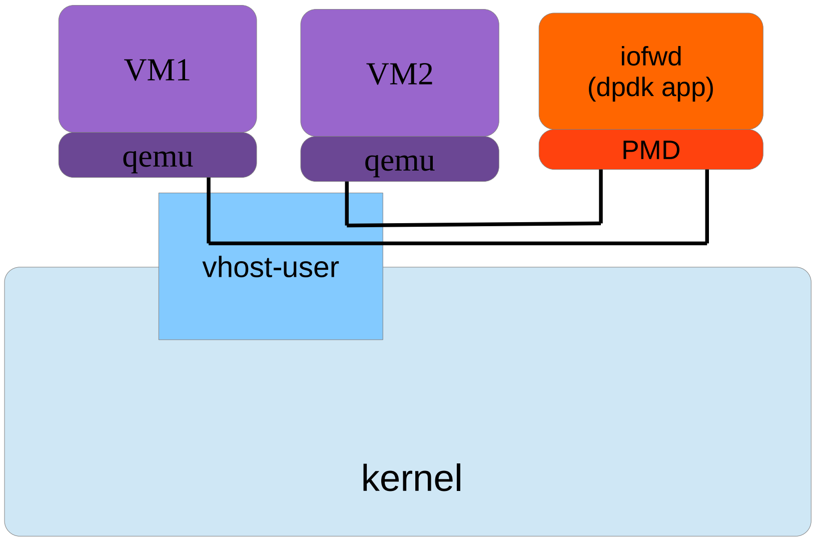
Make sure, iperf test session completes successfully and there are 0 missed/error/nombuf count in pmd stats.

To release ethernet port from namespace, execute below command.

```
ip netns exec tcp-server ip link set enP2p1s0f1 netns 1
```

### 3.1.4 Check two VMs communicating through vhost PMD

This test verifies whether two KVM virtual machines communicate through vhost PMD which is thin wrapper of DPDK userspace vhost library.



Setup instructions are given below. DPDK ships testpmd application as a utility to test DPDK libraries in accordance with its options. It is just like any other DPDK application and its usage has to be carefully understood and applied. Attempt on wrong parameter inputs would lead entire DPDK stack to crash. Refer to DPDK section for troubleshooting common mistakes.

```
# Register vhost driver in system using testpmd app
testpmd -c 0x0101 -m 1024 \
      -d /usr/lib/powerpc64le-linux-gnu/librte_pmd_vhost.so \
      --vdev 'eth_vhost1,iface=/tmp/sock1,queues=1' \
      --vdev 'eth_vhost2,iface=/tmp/sock2,queues=1'
      -- –portmask=0x3 -i
```

In the above testpmd command, we create two vhost sockets (and one receive queue in each port). You would need to create two VM instances (one on each vhost socket). On running testpmd with an interactive option ("-i"), it opens its own PMD shell to further run other commands (eg, display port stats, change port config etc). Without "-i", testpmd simply runs in packet forward mode.

As an additional note, testpmd option "--portmask" is similar to cpu core mask which informs testpmd application about what ports participate in packet forwarding operations (out of all detected PCIe devices).

Once vhost-user sockets are created, we need to set its ownership and access permission for the user we currently run qemu kvm guests.

```
# add user in kvm group if not already added
sudo adduser `whoami` kvm

#change ownership and permission of vhostuser sockets
sudo chgrp kvm /tmp/sock*
sudo chmod 0660 /tmp/sock*
```

Also, qemu is requested to use hugepages as memory backend for entire guest memory. This is required for virtio NIC in guest to work with DPDK vhost implementation. In this example, it is assumed that, guest is booting with 1024MB as its RAM as well. Given a reservation of already 8192MB we created in hugetlbfs, above guest RAM sizes are sufficiently subset of overall hugepage allocations. For any other guest OS, you might need to check with its hugepage consumption on booting time and avoid running out of memory accordingly.

Update libvirt xml for KVM guests using “virsh edit <kvm>” so as to add below numa related entries. Irrespective of guest cpu mode, guest NUMA topology has to be mentioned as below in order to enable shared memory mapping between these two KVM guests.

```
# For both VM1 and VM2 vCPUs, ensure hugepages backed memory is used
# and its memory mapping is shared.
  <memory unit='KiB'>1048576</memory>
  <currentMemory unit='KiB'>1048576</currentMemory>
  <memoryBacking>
    <hugepages>
      <page size='16384' unit='KiB' nodeset='0-1'/>
    </hugepages>
  </memoryBacking>

  <cpu>
    <topology … />
    <numa>
      <cell ... memAccess='shared'/>
      <cell ... memAccess='shared'/>
    </numa>
  </cpu>
```

```
# Add below entries in VM1 libvirt xml.
<interface type='vhostuser'>
  <mac address='52:54:c2:11:22:01'/>
  <source type='unix'
   path='/tmp/sock1'
   mode='client'/>
  <model type='virtio'/>
</interface>
```

```
# Add below entries in VM2 libvirt xml.
<interface type='vhostuser'>
  <mac address='52:54:c2:11:22:02'/>
  <source type='unix'
   path='/tmp/sock2'
   mode='client'/>
  <model type='virtio'/>
</interface>
```

In this test and also in other sections for virtualization scenarios between VMs, we cover below tests which is common across these scenarios (VM-VM), unless and until any specific instructions additionally passed to check virtual networking.

```
# Configure IP address on vNIC of VM1.
ifconfig eth0 10.0.0.11 up

# Configure IP address on vNIC of VM2.
ifconfig eth0 10.0.0.12 up
```

```
# Start iperf server (tcp) in VM1.
iperf -s

# Run netperf client in rest of the VMs.
iperf -c 10.0.0.11
```

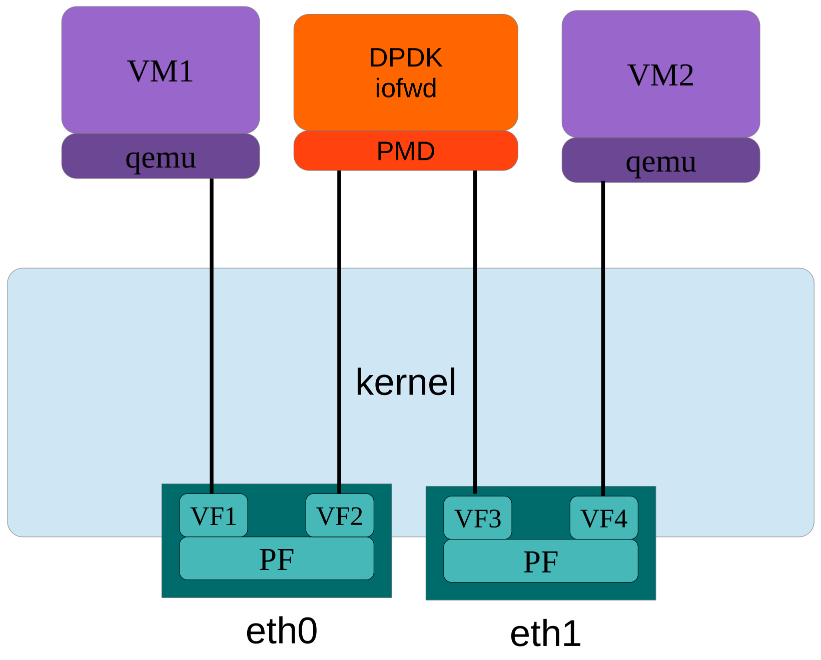
In above tests, TCP session is established between VM1 and VM2 and stream of packets exchanged between these two VMs. Verify whether DPDK dropped any packets. You can run below stats command in PMD shell (as mentioned in the beginning of this section).

```
# Report packets stat port 0. Use “all” instead of 0 for all ports

testpmd> show port stats 0

  ######################## NIC statistics for port 0
########################
  RX-packets: 306399     RX-missed: 0          RX-bytes:  18847534218
  RX-errors: 0
  RX-nombuf:  0
  TX-packets: 57930      TX-errors: 0          TX-bytes:  3823468

######################################################################
######
```

Make sure, iperf test session completes on both VM1 and VM2 and there are 0 missed/error/nombuf count in pmd stats.

Refer to known bugs in the end of this document.

### 3.1.5 Check two VMs communicating through SRIOV PMD



This test verifies whether DPDK could start poll mode driver on SRIOV Virtual Functions (VF) and there by, acting as packet forwarding application between two VMs on other VFs. In the figure above, eth0 and eth1 mean port 0 and port 1 of network adapter for the demonstration of this test.

Setup instructions are given below. It is assumed that, you have followed instructions as in Section 3.1.3 to create four virtual functions in your adapter and started two VMs where VM1 accessing VF1 in pci-passthrough and similarly VM2 accessing VF4 in pci-passthrough.

Also, ensure VF2 and VF3 are enabled for forwarding unmatched traffic (whose MAC addresses are those of VF inside VM).

```
# Mellanox CX4
# Add FDB entry for VF1 and VF4 over VF3 and VF2 respectively
sudo bridge fdb add <VF4 mac> dev <VF2 interface name>
sudo bridge fdb add <VF1 mac> dev <VF3 interface name>
```

In below setup instructions. testpmd uses EAL parameter “-w” which is to choose (i.e, whitelist) PCI devices for DPDK application to further configure PMD threads. As we need two ports for packet forwarding, we chose port mask 0x3 i.e out of detected ports 0,1,2,3,.. use port 0 and 1 for packet forwarding. Port mask is similar to CPU core mask as explained in Section 3.1.1.

```
# Start dpdk forward app on VF2 and VF3
testpmd -c 0x0101  \
      -w <VF2 PCI Addr> -w <VF3 PCI Addr>
      -- -i –portmask=0x3

# Start VM1 in pci-passthrough accessing VF1
<qemu cmd>

# Start VM2 in pci-passthrough accessing VF4
<qemu cmd>

# Enter “start” in testpmd interactive shell.
```

Run iperf tests as described in section 3.1.4.
Refer to known bugs in the end of this document.

### 3.1.6 Check ip_pipeline sample application for LPM, ACL vector libraries

This test verifies DPDK LPM and ACL vector libraries through ip_pipeline sample application which is shipped along with DPDK.

```
cd /usr/share/dpdk/examples/ip_pipeline
edit init.c (explained below)

source /usr/share/dpdk/dpdk-sdk-env.sh
make

./build/ip_pipeline -p 0x3 -f ./config/l3fwd.cfg
```

In init.c of ip_pipeline folder, set 3rd parameter to cpu_core_map_init() as “1” incase SMT is off. Otherwise, set it to be 8 which is the maximum number of SMT threads. 1st and 2nd parameters are the maximum number of Sockets and Cores.

Above sample l3fwd.cfg assumes your system CPU lcore 0 used as MASTER lcore to handle incoming and outgoing packets on DPDK Poll Mode Driver. System lcore 1 is used as ROUTING core to switch packets between TX and RX queues of PMD driver. If there is only one DPDK NIC (which has only two ports), you need to update its pktq_in and pktq_out variables to reflect current NIC configurations.

Also, you can add other EAL parameters to DPDK in the same config file under section “[EAL]” as shown below.

```
[EAL]
log_level = 2
socket_mem = 1024,1024
d = <librte_X_pmd.so>

[PIPELINE0]
type = MASTER
core = 0

[PIPELINE1]
type = ROUTING
core = 1
pktq_in = RXQ0.0 RXQ1.0
pktq_out = TXQ0.0 TXQ1.0 SINK0
encap = ethernet; encap = ethernet / ethernet_qinq / ethernet_mpls
ip_hdr_offset = 270
```

Once ip_pipeline application starts running, you will get its command shell “pipeline>”. You can add default route and forwarding rules for all packets (between port 0 and port 1) as below. Assuming packet generation is for a iperf session (say 10.4.4.3 ←-→ 10.4.4.4), we allow only those packets in subnet 10.4.4.0 and drop the rest by pushing into “SINK0”.

As in the above configuration file, RXQ0.0 and TXQ0.0 refers to port 0 - queue 0 of PMD, RXQ1.0 and TXQ1.0 refers to port 1 - queue 1 of PMD. SINK0 (port 2) is imaginary device within pipeline for the purpose of dropping packets.

```
# by default
pipeline> p 1 route add default 2 #SINK 0
pipeline> p 1 route add 10.4.4.4 32 port 1 ether 7c:fe:90:43:c2:1d
pipeline> p 1 route add 10.4.4.3 32 port 0 ether 7c:fe:90:43:c2:1c
pipeline> p 1 route ls
IP Prefix = 10.4.4.4/32 => (Port = 1, Next Hop HWaddress =
7c:fe:90:43:c2:1d)
IP Prefix = 10.4.4.3/32 => (Port = 0, Next Hop HWaddress =
7c:fe:90:43:c2:1c)
Default route: port 2 (entry ptr = 0x3efbf0bf9680)
pipeline>
```

As the above sample application does not forward ARP, ensure IP/MAC address is statically resolved in remote system running packet generator (iperf) as explained in section 3.1.3.

```
# set ip/mac in arp table
arp -s 10.4.4.4 7c:fe:90:43:c2:1d
ip netns exec tcp-server arp -s 10.4.4.3 7c:fe:90:43:c2:1c
```

Follow instructions on executing iperf as briefed in section 3.1.3. Once iperf completes, check the link stats in ip_pipeline application as below and verify if there are no missed/error packets.

```
# check link stats
pipeline> link ls
LINK0: flags=<UP>
      ether 7c:fe:90:43:c2:e4
      RX packets 1695586  bytes 2567091714
      RX errors 0  missed 0  no-mbuf 0
      TX packets 52295  bytes 3467694
      TX errors 0

LINK1: flags=<UP>
      ether 7c:fe:90:43:c2:e5
      RX packets 52328  bytes 3469854
      RX errors 0  missed 0  no-mbuf 0
      TX packets 1695568  bytes 2567068824
      TX errors 0

pipeline>
```

Ensure 0 RX/TX errors and Quit application using “quit” command.

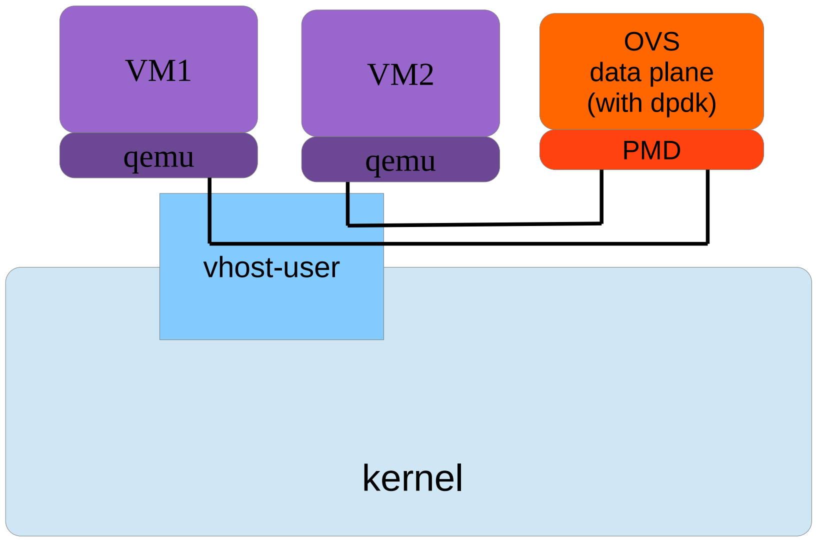
## 3.2 DPDK OVS tests

Objective of these tests are to test OpenVSwitch (OVS) dataplane handled by DPDK PMD threads. With DPDK accelerated data plane (instead of openvswitch.ko kernel module), OVS operates entirely in userspace. As recommended by OVS, it is required to reserve hugepages amounting 1GB before starting OVS vswitchd daemon. Refer Section 2 for instructions on booting system with hugepage parameter and starting DPDK service. Tests covered under OVS are:

1. Check two VMs communicating through DPDK OVS
2. Check two remote VMs communicating through DPDK OVS

By default, OVS daemon starts with data plane in kernel unless and until DPDK EAL parameters (cpu core mask, socket mem, pci ports etc) are registered in OVS database and OVS service is restarted. Due to OVS database entries, OVS processes can retain system configurations for DPDK (forwarding lcores, socket memory size etc) on system reboot.

### 3.2.1 Check two VMs communicating through DPDK OVS



This test verifies if two local VMs in a hypervisor can communicate in DPDK data plane using OVS bridge. Prior to setting up OVS bridge for Guest NIC and DPDK NIC, you need to pick ovs-vswitchd with DPDK data plane, instead of kernel data plane and initialize its EAL options.

```
# OVS DB setup and configurations
ovs-vsctl set Open_vSwitch . other_config:dpdk-init=true
ovs-vsctl set Open_vSwitch . other_config:dpdk-lcore-mask=0x10101
ovs-vsctl set Open_vSwitch . other_config:dpdk-socket-mem=1024
ovs-vsctl set Open_vSwitch . other_config:dpdk-extra="
    -d  /usr/lib/powerpc64le-linux-gnu/librte_pmd_vhost.so
    -d /usr/lib/powerpc64le-linux-gnu/librte_pmd_mlx5.so
    --vhost-owner libvirt-qemu:kvm --vhost-perm 0660"

# Restart OVS service
systemctl restart openvswitch-switch.service
```

Above configuration instructions are common across all OVS usecases with DPDK (as in following sections) unless specifically mentioned for any additional configuration changes.

```
# Setup OVS bridge and VM ports
ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev

ovs-vsctl add-port br0 vhost-user-1 \
      -- set interface vhost-user-1 type=dpdkvhostuser

ovs-vsctl add-port br0 vhost-user-2 \
      -- set interface vhost-user-2 type=dpdkvhostuser
```

Default flow table rule for newly added ports in OVS bridge is to learn MAC and forward packets like Linux bridge. Once ports are added, PMD threads for these ports would be kicked off which can be confirmed through couple of ovs-appctl commands to query its configuration and stats.

```
# query PMD info
ovs-appctl dpif-netdev/pmd-stats-show

main thread:
…
pmd thread numa_id 0 core_id 13:
      emc hits:0
      megaflow hits:0
      avg. subtable lookups per hit:0.00
      miss:0
      lost:0
      polling cycles:105641538703 (100.00%)
      processing cycles:0 (0.00%)
```

```
# query port rx queue IDs
ovs-appctl dpif-netdev/pmd-rxq-show

pmd thread numa_id 0 core_id 13:
      isolated : false
      port: vhost-user-1      queue-id: 0
      port: vhost-user-2      queue-id: 0
```

As shown in pmd-rxq-show, there is one rx queue per port created and PMD thread for these queues is executed in core 13.

As in Section 3.1.4, ensure KVM libvirt XML is updated for hugepages backed guest memory and added vhostuser backed virtio vNIC.

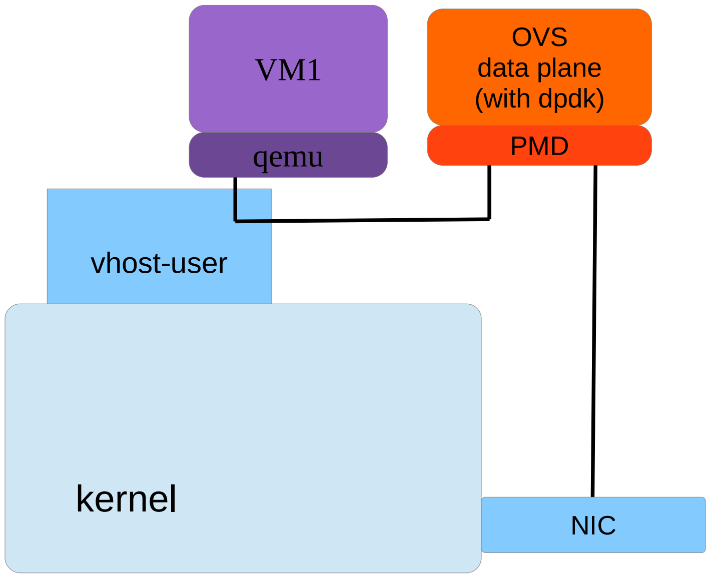
```
# For both VM1 and VM2 vCPUs, ensure hugepages backed memory is used
# and its memory mapping is shared.
  <memory unit='KiB'>1048576</memory>
  <currentMemory unit='KiB'>1048576</currentMemory>
  <memoryBacking>
    <hugepages>
      <page size='16384' unit='KiB' nodeset='0-1'/>
    </hugepages>
  </memoryBacking>

  <cpu>
    <topology … />
    <numa>
      <cell ... memAccess='shared'/>
      <cell ... memAccess='shared'/>
    </numa>
  </cpu>
```

```
# Add below entries in VM1 libvirt xml.
<interface type='vhostuser'>
  <mac address='52:54:c2:11:22:01'/>
  <source type='unix'
   path='/var/run/openvswitch/vhost-user-1'
   mode='client'/>
  <model type='virtio'/>
</interface>

# in same way, start VM2 using vhost-user-2, different MAC address.
<interface type='vhostuser'>
  <mac address='52:54:c2:11:22:02'/>
  <source type='unix'
   path='/var/run/openvswitch/vhost-user-2'
   mode='client'/>
  <model type='virtio'/>
</interface>
```

Run iperf tests as described in section 3.1.4. Instead of using testpmd stats, “ovs-appctl dpif-netdev/pmd-stats-show” command can be used to study DPDK data path statistics.

Refer to known bugs in the end of this document.

### 3.2.2 Check two remote VMs communicating through DPDK OVS



This test verifies if two VMs can communicate each other across hypervisors. In each hypervisor, OVS DB and vswitchd configurations have to be done as in Section 3.2.1. Additionally, follow below instructions in each host to configure ovs bridges.

```
# Setup OVS integration bridge and attach VM port
ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev

ovs-vsctl add-port br0 usvhost \
      -- set interface usvhost type=dpdkvhostuser ofport_request=1
ovs-vsctl add-port br0 tep – set interface tep type=vxlan \
      options:remote_ip=<remote host IP> ofport_req=10

# Setup OVS external bridge, attach DPDK port and configure IP address
ovs-vsctl add-br br1 -- set bridge br1 datapath_type=netdev
ovs-vsctl add-port br1 dpdk0 \
      –- set interface dpdk0 type=dpdk ofport_request=1
ifconfig br1 <local host IP>/24
```

```
# Update OVS flow table rules for integration bridge br0
ovs-ofctl del-flows br0
ovs-ofctl add-flow br0 priority=5,ip,nw_dst=$(VM1_IP},action=output:1
ovs-ofctl add-flow br0 priority=5,ip,nw_dst=$(VM2_IP},action=output:10
ovs-ofctl add-flow br0 priority=0,action=NORMAL

# Update OVS flow table rules for external bridge br1
ovs-ofctl del-flows br1
ovs-ofctl add-flow br1 priority=5,ip,in_port=LOCAL,action=output:1
ovs-ofctl add-flow br1 priority=5,ip,in_port=1,action=output:LOCAL
ovs-ofctl add-flow br1 priority=0,action=NORMAL
```

In above instructions, “usvhost” is name of virtual port connected to VM1. “tep” is name of tunnel end point whose IP address is nothing but same IP address the phy NIC have been using from your network, prior to setting up DPDK. “br0” is integration bridge and “br1” is external bridge. “VM1_IP” is IP address that VM1 OS has assigned to its vNIC. Similarly, “VM2_IP” is IP address that VM2 OS has assigned to its vNIC. On executing above instructions in remote hypervisor (where VM2 is local to it), swap VM1 and VM2 IP addresses in above steps.

As in section 3.2.1, start a VM in each hypervisor with mentioned memory and network device configurations. As we are using vxlan overlay between hypervisors (refer “type” in ovs add-port command), reduce MTU size in each VM so as to accommodate guest ethernet traffic within VxLAN.

```
# Inside each VM, Reduce MTU of guest virtio to 1450
ifconfig eth0 mtu 1450
```

Run iperf tests as described in section 3.1.4.
Refer to known bugs in the end of this document.

# 4. Troubleshooting

Listed below are possible common issues you may come across on setting up and running DPDK software.

## 4.1 Generic issues

| Problem | Cause | Solution |
|---|---|---|
| iperf connect failed: No route to host | firewall in iperf server node could be blocking new iperf session. | stop firewall by running:<br>systemctl stop firewalld.service |

## 4.2 DPDK issues

| Problem | Cause | Solution |
|---|---|---|
| EAL panic – cannot set affinity | You may be passing an invalid lcore or not supported lcore number. | Run “lscpu” and infer lcore from on-line CPUs.<br>If using -c, convert lcore number into CPU bitmask. Eg. lcore 0,8 and 16 translates to 0x010101.<br>If using -l, pass lcore numbers as they are. Eg. -l 0,8,16<br>This version of DPDK supports only upto 256 lcores in ppc64le architecture. Contact LTC Networking to address any limit more than it. |
| EAL panic – lcore empty set | You may be passing only one lcore | At a minimum, two lcores needs to be given to EAL to start DPDK. Choose two online CPUs from “lscpu” and pass them to EAL. |
| EAL panic – cannot get hugepage information | Either hugetlbfs is not mounted or there is insufficient (or zero) hugepages reserved. | Check if hugetlbfs mounted ( /proc/mounts), if not run:<br>mount -t hugetlbfs nodev /mnt/huge<br><br>Check /proc/meminfo for Hugepages total, free sections.<br><br>Run“sysctl -w vm.nr_hugepages=N” to add N*16MB in hugetlbfs. |
| PMD librte_pmd_mlx5: cannot list devices, is ib_uverbs loaded? | Mellanox libraries not loaded correctly. | Run /etc/init.d/openibd start |
| EAL exit error 1 – portmask is not consistent to port ids | Mentioned portmask is not matching with detected ports. | Check passing all detected ports. |

| Problem | Cause | Solution |
|---|---|---|
| EAL exit error 1 – cannot create new DMA window, error 22 | There is insufficient hugepages available. | Check /proc/meminfo for Hugepages total, free sections and ensure, minimum no of pages reserved as suggested in Section 2. |
| PMD i40evf_dev_configure(): VF can't disable HW CRC Strip | CRC strip enabled by default | Use –crc-strip in parameters for application (testpmd) |
| PMD i40e port reporting Link Down | Unqualified network cable may be in use. | Ensure QSFP+ network cable connected to XL710 adapter is in support matrix: http://www.intel.ph/content/dam/www/public/us/en/documents/product-briefs/ethernet-qsfp-cables-brief.pdf |
| EAL no driver found for net_pcap_rx_0 | pcap pmd not turned on in configuration | Recompile source with CONFIG_RTE_LIBRTE_PMD_PCAP=y |

## 4.3 OpenVSwitch issues

| Problem | Cause | Solution |
|---|---|---|
| ovs-vsctl: Port dpdk0 showing error "could not open network device dpdk0 (Address family not supported by protocol)" | OVS daemon currently running is not the one with DPDK data plane enabled. | Ensure DPDK enabled OVS service up and running. In case of Ubuntu, you would need to use "update-alternatives" command (when syslog reports "DPDK not supported in this copy of Open vSwitch". |
| ovs-vsctl: Port dpdk0 showing error "could not open network device dpdk0 (Cannot allocate memory)" | DPDK could not find local numa memory for a PCI device detected. | If PMD has to run on all DPDK ports, make sure your system has local numa memory for each PCI device detected by DPDK. Refer "2. Product setup and configurations" Section.<br><br>Below instructions for your numa topology:<br>1. Note down numa ID in /sys/bus/pci/devices/<PCIaddr>/numa_node<br>2. Request ovs-vswitchd to use local mem for dpdk port by "--dpdk--socket-mem=ID1_mem,ID2_mem" |

## 4.4 Virtualization issues

| Problem | Cause | Solution |
|---|---|---|
| qemu failed initializing vhost-user memory map | Guest memory backing is not from hugetlbs | Add "-object memory-backend-file,id=mem,size=<RAM MB>,mem-path=<hugetlbfs mount>,share=on -mem-prealloc |

## 4.5 SRIOV issues

| Problem | Cause | Solution |
|---|---|---|
| PMD thread fails to start on VF port | LTC bug 143855 | Fixed in MLNX OFED 3.4 |

## 4.6 Application issues

| Problem | Cause | Solution |
|---|---|---|
| pipeline aborted - Cannot find LINK2 for RXQ2.0 | Application configuration expects port 3 ,but it is not present | Update config file used by your application to reflect correct number of ports in any PIPELINE section. |
| pipeline aborted – Initializing LINK0 (0) (1 RXQ, 1 TXQ) panic | Insufficient or empty memory configuration for socket associated with NIC in PCI bus | Update config file used by your application to reflect sufficient memory for all socket. Eg, socket_mem = 1024,1024 |
| pipeline aborted - LINK0 pci_bdf is not configured (port_mask is not provided) | | |

# 5. Known issues

Listed below are possible common issues you may come across on setting up and running DPDK software.

| Problem | Cause | Solution |
|---|---|---|
| 3.1.3: Vm-Vm not connecting each other (arp-reply not received) | LTC bug 151082<br>(isolated VF needs more validation) | Under investigation |
| 3.1.5: Vm-Vm not connecting each other (arp-reply not received) | LTC bug 151082<br>(isolated VF needs more validation) | Under investigation |
| 3.2.2: iperf (or any tcp) session would not establish between two kvm guests | LTC bug 147080<br><br>Due tcp checksum offload that is turned on for virtio nic (by default ON), invalid checksum passed on the other end and TCP stack rejects SYN. | Not reproducible in ubuntu 17.04 with libvirt default settings of tx/rx offload for vhost backend. |
| i40e txq/rxq stats 0 in dpdk-procinfo | Queue stats not supported in XL710 as per i40e_dev_queue_stats_mapping() | N/A |