Intel® Data Plane Development Kit - L3 Forwarding in a Virtualization Environment Sample Application

User Guide

April 2012

Intel Confidential

Reference Number: 496040-1.0

Contents

Revision History

Date	Revision	Description
April 2012	1.0	Initial release

1.0 Introduction

The L3 Forwarding in a Virtualization Environment sample application is a simple example of packet processing using the Intel® Data Plane Development Kit (Intel® DPDK). The application performs L3 forwarding that takes advantage of Single Root I/O Virtualization (SR-IOV) features in a virtualized environment.

1.1 Documentation Roadmap

The following is a list of Intel® DPDK documents in suggested reading order:

- **Release Notes**: Provides release-specific information, including supported features, limitations, fixed issues, known issues and so on. Also, provides the answers to frequently asked questions in FAQ format.
- **Getting Started Guide**: Describes how to install and configure the Intel® DPDK software; designed to get users up and running quickly with the software.
- **Programmer's Guide**: Describes:
  - The software architecture and how to use it (through examples), specifically in a Linux* application (linuxapp) environment
  - The content of the Intel® DPDK, the build system (including the commands that can be used in the root Intel® DPDK Makefile to build the development kit and an application) and guidelines for porting an application
  - Optimizations used in the software and those that should be considered for new development

  A glossary of terms is also provided.
- **API Reference**: Provides detailed information about Intel® DPDK functions, data structures and other programming constructs.
- **Sample Application User Guides**: A set of guides, each describing a sample application that showcases specific functionality, together with instructions on how to compile, run and use the sample application.

2.0 Overview

The application demonstrates the use of the `hash` and `LPM` libraries in the Intel® DPDK to implement packet forwarding. The initialization and run-time paths are very similar to those of the L2 forwarding application (see the *Intel® DPDK L2 Forwarding in a Virtualization Environment Sample Application User Guide* for more information). This guide highlights the differences between the two applications. The main difference from the L2 Forwarding in a Virtualization Environment sample application is that the forwarding decision is taken based on information read from the input packet.

The lookup method is either hash-based or LPM-based and is selected at compile time. When the selected lookup method is hash-based, a hash object is used to emulate the flow classification stage. The hash object is used in correlation with the flow table to map each input packet to its flow at runtime.

The hash lookup key is represented by the DiffServ 5-tuple composed of the following fields read from the input packet: Source IP Address, Destination IP Address, Protocol, Source Port and Destination Port. The ID of the output interface for the input packet is read from the identified flow table entry. The set of flows used by the application is statically configured and loaded into the hash at initialization time. When the selected lookup method is LPM based, an LPM object is used to emulate the forwarding stage for IPv4 packets. The LPM object is used as the routing table to identify the next hop for each input packet at runtime.

The LPM lookup key is represented by the Destination IP Address field read from the input packet. The ID of the output interface for the input packet is the next hop returned by the LPM lookup. The set of LPM rules used by the application is statically configured and loaded into the LPM object at the initialization time.

Note: Please refer to the “Virtual Function Setup Instructions” in the *Intel® DPDK L2 Forwarding in a Virtualization Environment Sample Application User Guide* for virtualized test case setup.

3.0 Compiling the Application

To compile the application:

1. Go to the sample application directory:

```
export RTE_SDK=/path/to/rte_sdk
cd ${RTE_SDK}/examples/l3fwd-vf
```

2. Set the target (a default target is used if not specified). For example:

```
export RTE_TARGET=x86_64-default-linuxapp-gcc
```

   See the *Intel® DPDK Getting Started Guide* for possible `RTE_TARGET` values.

3. Build the application:

```
make
```

Note: The compiled application is written to the `build` subdirectory. To have the application written to a different location, the `O=/path/to/build/directory` option may be specified in the `make` command.

4.0 Running the Application

The application has a number of command line options:

```
./build/l3fwd-vf [EAL options] -- -p PORTMASK
[--config(port,queue,lcore)[,(port,queue,lcore]]
```

where,

- `--p PORTMASK`: Hexadecimal bitmask of ports to configure
- `--config (port,queue,lcore)[,(port,queue,lcore]`: determines which queues from which ports are mapped to which cores

For example, consider a dual processor socket platform where cores 0,2,4,6, 8, and 10 appear on socket 0, while cores 1,3,5,7,9, and 11 appear on socket 1. Let's say that the programmer wants to use memory from both NUMA nodes, the platform has only two ports and the programmer wants to use two cores from each processor socket to do the packet processing.

To enable L3 forwarding between two ports, using two cores from each processor, while also taking advantage of local memory accesses by optimizing around NUMA, the programmer must enable two queues from each port, pin to the appropriate cores and allocate memory from the appropriate NUMA node. This is achieved using the following command:

```
./build/l3fwd-vf -c 0x00f -n 3 -- -p 0x3 --config="(0,0,0),(0,1,2),(1,0,1),(1,1,3)"
```

In this command:

- The `-c` option enables cores 0, 1, 2, 3
- The `-p` option enables ports 0 and 1
- The `--config` option enables two queues on each port and maps each (port,queue) pair to a specific core. Logic to enable multiple RX queues using RSS and to allocate memory from the correct NUMA nodes is included in the application and is done transparently. The following table shows the mapping in this example:

Port	Queue	lcore	Description
0	0	0	Map queue 0 from port 0 to lcore 0.
0	1	2	Map queue 1 from port 0 to lcore 2.
1	0	1	Map queue 0 from port 1 to lcore 1.
1	1	3	Map queue 1 from port 1 to lcore 3.

Refer to the *Intel® DPDK Getting Started Guide* for general information on running applications and the Environment Abstraction Layer (EAL) options.

Note: This `l2fwd-vf` application uses the Virtual Function Pole Mode Driver (PMD) and it has only one RX and one TX queue. So while giving the `–config` option, it is necessary to ensure that there is a one-to-one mapping between RX queue/port and queue. For example “(0,0,0), (0,0,1)” results in an error statement, while “(0,0,0), (1,0,1)” or “(0,0,0), (1,0,0)” works because the user refers to the RX queue 0 of port 0 and port 1 in respective entries.

5.0 Explanation

The operation of this application is similar to that of the basic L3 Forwarding Sample Application. See the “Explanation” section in the *Intel® DPDK L3 Forwarding Sample Application User Guide* for more information.

§ §