Intel® Ethernet Controller E810 Data Plane Development Kit (DPDK) 22.11/23.03

Configuration Guide

NEX Cloud Networking Group (NCNG)

Rev. 1.1

June 2023

Doc. No.: 764257, Rev.: 1.1

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

This document (and any related software) is Intel copyrighted material, and your use is governed by the express license under which it is provided to you. Unless the license provides otherwise, you may not use, modify, copy, publish, distribute, disclose or transmit this document (and related materials) without Intel's prior written permission. This document (and related materials) is provided as is, with no express or implied warranties, other than those that are expressly stated in the license.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors which may cause deviations from published specifications.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

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

Copyright © 2023, Intel Corporation. All rights reserved.

Did this document help answer your questions?

Contents

Did this document help answer your questions?

Did this document help answer your questions?

Revision History

Revision	Date	Comments
1.1	June 16, 2023	Changes include: • Added 23.03. • Added new chapter, Advanced Features and Debug.
1.0	March 7, 2023	Initial public release.

Did this document help answer your questions?

1.0 Introduction

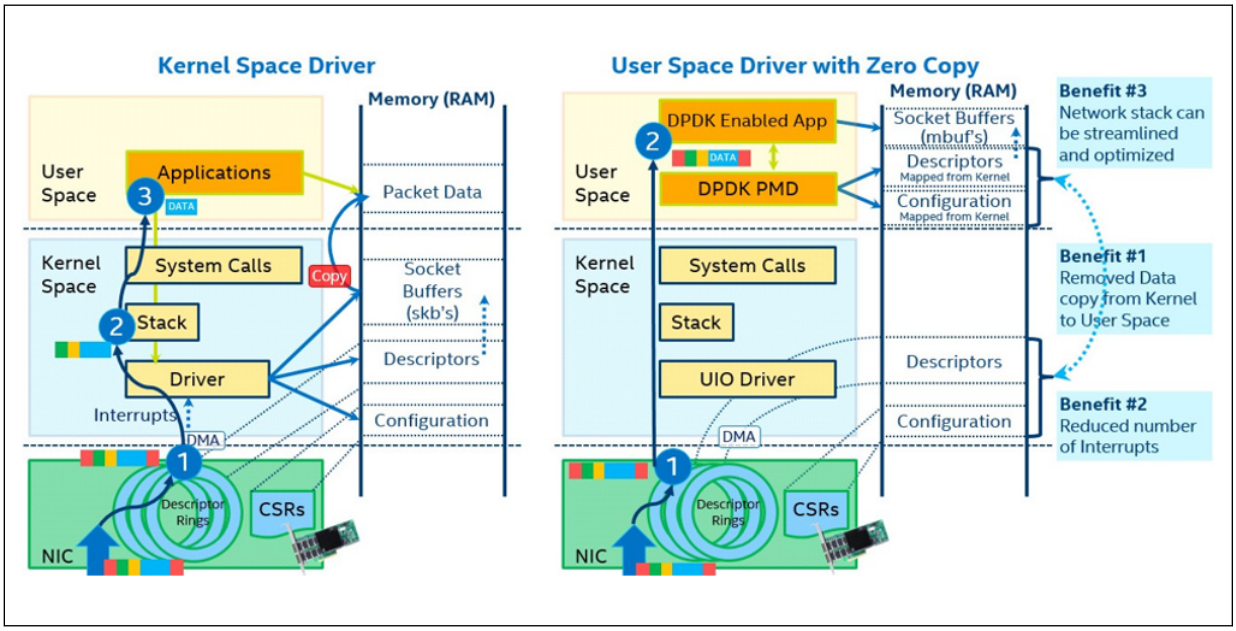
This document is designed to provide instructions for configuring and testing Intel® Ethernet 800 Series Network Adapters with Data Plane Development Kit (DPDK).

1.1 DPDK Overview

DPDK is a set of libraries and drivers that perform fast packet processing. This enables a user to create optimized performance with packet processing applications.

DPDK bypasses the OS network stack, avoiding the associated latency, and maps hardware registers to user space. A DPDK-enabled application processes packets faster by allowing NICs to DMA packets directly into an application's address space and having the application poll for packets, thereby avoiding the overhead of interrupts from the NIC.

Figure 1. Packet Processing Kernel Space vs. User Space



Did this document help answer your questions?

Key components of DPDK include the following:

- **Environment Abstraction Layer (EAL)** — Provides a generic interface that hides the environment specifics from the application and libraries.
- **Ethernet Poll Mode Driver (PMD)** — Designed to work without asynchronous, interrupt-based signaling mechanisms.
- **Memory management** — Allocates pools of objects in memory created in huge page memory space, uses a ring to store free objects, and spreads objects evenly across DRAM channels to optimize access speed.
- **Buffer management** — Pre-allocates fixed-size buffers that are stored in memory pools, significantly reducing the amount of time spent by the operating system allocating and deallocating buffers.
- **Queue management** — Replaces spinlocks with safe lockless queues, allowing different software components to process packets while avoiding unnecessary wait times.
- **Flow classification** — Improves throughput by implementing Intel® Streaming SIMD Extensions (Intel® SSE) to produce a hash based on tuple information, enabling packets to be placed into flows quickly for processing.
- **Packet Forwarding Algorithm Support** — Includes Hash (librte_hash) and Longest Prefix Match (LPM, librte_lpm) libraries to support the corresponding packet forwarding algorithms.

DPDK is composed of several directories:

- ***lib*** - Source code of DPDK libraries
- ***drivers*** - Source code of DPDK poll-mode drivers
- ***app*** - Source code of DPDK applications (automatic tests)
- ***examples*** - Source code of DPDK application examples
- ***config, buildtools, mk*** - Framework-related makefiles, scripts, and configuration

Refer to dpdk.org for more details.

Did this document help answer your questions?

1.2 Known Issues

- With the current *ice* PF driver, there might not be a way for a trusted DPDK VF to enable unicast promiscuous without turning on `ethtool --priv-flags` with `vf-true-promic-support`.
- If a VLAN with an Ethertype of 0x9100 is configured to be inserted into the packet on transmit, and the packet, prior to insertion, contains a VLAN header with an Ethertype of 0x8100, then the 0x9100 VLAN header is inserted by the device after the 0x8100 VLAN header. The packet is transmitted by the device with the 0x8100 VLAN header closest to the Ethernet header
- For the Intel® Ethernet 800 Series adapter in 8-port, 10 Gb configuration, the device might generate errors such as shown in the example below on Linux PF or VF driver load due to RSS profile allocation. Ports that report this error will experience RSS failures resulting in some packet types not being properly distributed across cores.

  dmesg: VF add example:

  ```
  ice_add_rss_cfg failed for VSI:XX, error:ICE_ERR_AQ_ERROR
  VF 3 failed opcode 45, retval: -5
  ```

  DPDK v22.11 testpmd example:

  ```
  Shutting down port 0...
  Closing ports...
  iavf_execute_vf_cmd(): No response or return failure (-5) for cmd 46
  iavf_add_del_rss_cfg(): Failed to execute command of OP_DEL_RSS_INPUT_CFG
  ```

  — **Workaround:** Disable RSS using the `--disable-rss` flag when starting DPDK. Afterwards, only enable the specific RSS profiles that are needed.

Did this document help answer your questions?

2.0 DPDK Requirements

2.1 System Requirements

For DPDK system requirements, refer to Section 2 of the following document:

http://doc.dpdk.org/guides/linux_gsg/

2.2 Software/Firmware Requirements

- Operating Systems:
  - RHEL 8.6
  - RHEL 9
  - Ubuntu 20.04.5 LTS
  - Ubuntu 22.04.1 LTS
  - Ubuntu 22.10
  - SLES 15 SP4
- Linux Kernel: 5.15.0-46-generic

The following table lists the driver, firmware, and package versions recommended for use with the supported DPDK version.

Table 1. DPDK Recommended Matching List

DPDK	Software Release	*ice* Kernel Driver	*iavf* Kernel Driver	NVM Version	Firmware	DDP OS Package	DDP Comms Package	DDP Wireless Edge Package
20.05	25.2	1.0.4	4.0.1	2.00	1.4.1.13	1.3.13.0	1.3.17.0	N/A
20.08	25.3 25.4	1.1.4	4.0.1	2.10 / 2.12 2.15 / 2.14	1.5.1.5/1.5 .1.9	1.3.16.0	1.3.20.0	N/A
20.08 / 20.11[1]	25.5	1.21	4.0.1	2.20 / 2.22	1.5.2.8	1.3.18.0	1.3.22.0	N/A
20.11[1] / 21.02	25.6	1.3.2	4.0.2	2.30 / 2.32	1.5.3.7	1.3.20.0	1.3.24.0	N/A
	26.1	1.4.11	4.1.1	2.40 / 2.42	1.5.4.5	1.3.24.0	1.3.28.0	1.3.4.0
21.02[1] / 21.05	26.3	1.5.8	4.1.1	2.50 / 2.52	1.5.5.6	1.3.26.0	1.3.30.0	1.3.6.0
21.05 / 21.08[1] / 21.11[1]	26.4	1.6.4 / 1.6.7	4.2.7	3.00 / 3.02	1.6.0.6	1.3.26.0	1.3.30.0	1.3.6.0
21.11	26.8	1.7.16	4.3.19	3.10 / 3.12	1.6.1.9	1.3.27.0	1.3.31.0	1.3.7.0
21.11[1] / 22.03	27.1	1.8.3	4.4.2	3.20 / 3.22	1.6.2.9	1.3.28.0	1.3.35.0	1.3.8.0
22.03 / 22.07[1]	27.5	1.9.11	4.5.3	4.00 / 4.02	1.7.0.7	1.3.30.0	1.3.37.0	1.3.10.0

continued...

Did this document help answer your questions?

DPDK	Software Release	*ice* Kernel Driver	*iavf* Kernel Driver	NVM Version	Firmware	DDP OS Package	DDP Comms Package	DDP Wireless Edge Package
22.07[1]	27.7	1.10.1.2	4.6.1	4.10 / 4.12	1.7.1.7	1.3.30.0	1.3.37.0	1.3.10.0
22.07 / 22.11 / 23.03	28.0	1.11.14	4.8.2	4.20/4.22	1.7.2.4	1.3.30.0	1.3.40.0	1.3.10.0
Note: 1. Compatibility testing (basic use case testing including VF).								

2.2.1 Updating the NVM with a DPDK Driver

If all of the following are true:

- You want to update or inventory the device based on the Intel® Ethernet 800 Series.
- You are using the DPDK driver.
- The *ice* device driver is not bound to any port on the device.

Then you must:

1. Bind the kernel driver to the device.
   a. Make sure the *ice* kernel driver is installed.
   b. Use **lspci** to discover the PCI location of the device port you want to update/inventory (in <Bus:Device.Function> format (for example, 04:00.0))
   c. Bind the port with the kernel driver:

```
# usertools/dpdk-devbind.py -b <i40e|ice> <B:D.F>
```

2. Download the appropriate version of the NVM Update Utility from the Intel Support site: https://www.intel.com/content/www/us/en/search.html#q=e810%20nvm%20update&sort=relevancy
3. Run **nvmupdate**.
   - NVM update example:

```
# nvmupdate -u -l -c nvmupdate.cfg
```

   - NVM inventory example:

```
# nvmupdate -i -l -c nvmupdate.cfg
```

4. Reboot the system.
5. Restore your initial driver configuration by loading the DPDK driver.

```
# usertools/dpdk-devbind.py -b vfio_pci <B:D.F>
```


Did this document help answer your questions?

3.0 DPDK Installation and Configuration

To run a DPDK application, some customization might be required on the target machine.

For more details, refer Section 2.3 of the following document:

http://doc.dpdk.org/guides/linux_gsg/

NOTE

The configuration below has been tested on Ubuntu OS with kernel 5.15.0-46-generic.

3.1 System Configuration

For the best performance, refer to the reference system configuration details listed in the latest *Intel NIC Performance Report* hosted on DPDK.org at:

http://core.dpdk.org/perf-reports/

3.2 BIOS Settings

Power Management → CPU Power and Performance Policy <Performance>

Intel processors have a power management feature where the system goes in power savings mode when it is being underutilized. This feature should be turned off to avoid variance in performance. The system should be configured for maximum performance (BIOS configuration). The downside is that even when the host system is idle, the power consumption is not down.

```
Power and Performance → CPU Power and Perf Policy → Performance
Power and Performance → Workload Configuration→ I/O Sensitive
```

For maximum performance, low-power processor states (C6, C1 enhanced) should be disabled:

```
CPU C-state Disabled
```

P-States, which are designed to optimize power consumption during code execution, should be disabled:

```
CPU P-state Disabled
```


Did this document help answer your questions?

or:

```
Advanced → Power & Performance → CPU C State Control → Package C-State=C0/C1
State
Advanced → Power & Performance → CPU C State Control → C1E=Disabled
```

Turboboost/Speedstep

Speedstep is a CPU feature that dynamically adjusts the frequency of processor to meet processing needs, decreasing the frequency under low CPU-load conditions. Turboboost over-clocks a core when the demand for CPU is high. Turboboost requires that Speedstep is enabled.

These two configurations could introduce a variance in data plane performance when there is a burst of packets. For consistency of behavior, these two features should be disabled.

```
Enhanced Intel® Speedstep® Tech Disabled
Turbo Boost Disabled
```

Virtualization Extensions

Intel virtualization extensions, Intel® Virtualization Technology (Intel® VT) and Intel® Virtualization Technology for Directed I/O (Intel® VT-d), and DMA remapping (DMAR) must be turned on. VT-d enables IOMMU virtualization capabilities that are required for PCIe pass-through. Also, interrupt remapping should be enabled so that hardware interrupts can be remapped to a VM for PCIe pass-through.

Enable these extensions through the platform's BIOS settings.

```
Intel VT For directed I/O(VT-d) Enabled
Intel Virtualization Technology (VT-x) Enabled
```

Hyperthreading

Hyperthreading is Intel's simultaneous multi-threading technology For each physical processor core that is present, the operating system addresses two virtual (logical) cores and shares the workload between them when possible. Each logical core shares the resources (L1 and L2 cache, registers) of the physical core. This is controlled by a setting in the BIOS.

In general, data plane performance suffers when hyperthreading is enabled. Therefore, the recommendation is to disable it.

Hyperthreading configuration is a BIOS setting, and changing it requires a reboot.

If hyperthreading is enabled, it is still possible to obtain the same performance as with hyperthreading disabled. To do this, isolate the extra logical cores (see CPU isolation) and do not assign any threads to them.

```
Processor Configuration → Hyper-threading → Enabled for Throughput/Disabled for
Latency
```


Did this document help answer your questions?

3.3 Hugepages Setup

Hugepage support is required for the large memory pool allocation used for packet buffers. By using hugepage allocations, performance is increased since fewer pages are needed, and therefore less Translation Lookaside Buffers (TLBs, high speed translation caches), which reduce the time it takes to translate a virtual page address to a physical page address. Without hugepages, high TLB miss rates would occur with the standard 4K page size, slowing performance.

For 1 GB pages

It is not possible to reserve the hugepage memory after the system has booted. The size must be specified explicitly and can also be optionally set as the default hugepage size for the system.

The 1 GB hugepage option can be added in Grub along with IOMMU in kernel command line, as shown in High Performance of Small Packets on 100G NIC- Use 16 Bytes Rx Descriptor Size.

To reserve 4 GB of hugepage memory in the form of four 1 GB pages, the following options should be passed to the kernel:

```
default_hugepagesz=1G hugepagesz=1G hugepages=4
```

Once the hugepage memory is reserved, to make the memory available for DPDK use, execute the following:

```
mkdir /mnt/huge
mount -t hugetlbfs nodev /mnt/huge
```

For 1 GB pages, the mount point can be made permanent across reboots, by adding the following line to the */etc/fstab* file:

```
nodev /mnt/huge hugetlbfs pagesize=1GB 0 0
```

NOTE

There are multiple ways to create hugepages under RHEL 7.x, following is one example that shows the steps to create four pages of 1 GB:

1. Create a mounting point for huge pages with auto-mount.

   ```
   cd /mnt
   mkdir huge
   ```

2. Modify */etc/fstab* to include:

   ```
   nodev /mnt/huge hugetlbfs  pagesize=1GB 0 0
   ```

3. Modify and update grub to set up hugepages.

   ```
   # /etc/default/grub
   GRUB_CMDLINE_LINUX="default_hugepagesz=1G hugepagesz=1G hugepages=4"
   ```


Did this document help answer your questions?

For RHEL:

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

For Ubuntu:

```
Update-grub
```

4. Reboot the system and check Huge Page allocation.

```
# cat /proc/meminfo | grep HugePages_Total
HugePages_Total:       4
# cat /proc/meminfo | grep Hugepagesize
Hugepagesize:    1048576 kB
```

For 2 MB pages

Hugepages can be allocated after the system has booted. This is done by echoing the number of hugepages required to a nr_hugepages file in the */sys/devices/* directory.

For a single-node system, the command to use is as follows (assuming that 1024 pages are required):

```
echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
mount -t hugetlbfs nodev /mnt/huge
```

On a NUMA machine, pages should be allocated explicitly on separate nodes:

```
mkdir -p /mnt/huge
echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/
nr_hugepages
echo 1024 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/
nr_hugepages
mount -t hugetlbfs nodev /mnt/huge
```

3.4 IOMMU

In addition, to run the DPDK with Intel® VT-d, the `iommu=pt` kernel parameter must be used. This results in pass-through of the DMA Remapping (DMAR) lookup in the host. Also, if *INTEL_IOMMU_DEFAULT_ON* is not set in the kernel, the `intel_iommu=on` kernel parameter must be used as well. This ensures that Intel IOMMU is initialized as expected.

NOTE

While using `iommu=pt` is compulsory for *igb_uio* driver, the vfio-*pci* driver can work with both `iommu=pt` and `iommu=on`.

3.5 CPU Isolation

isolcpus is one of the kernel boot parameters that isolates certain CPUs from kernel scheduling, which is especially useful if you want to dedicate some CPUs for special tasks with the least amount or unwanted interruption (but cannot get to 0) in a multi-core system.

Did this document help answer your questions?

3.6 RCU Callbacks

To eliminate local timer interrupts, RCU callbacks need to be isolated as well. This is done either in the kernel config, or by the **rcu_nocbs** grub option.

3.7 Tickless Kernel

For high-performance applications, using a tickless kernel can result in improved performance. The host kernel must have the cores operating in tickless mode, and the same cores should be dedicated to the application.

The host kernel might have been built with the CONFIG_NO_HZ_FULL_ALL option. If so, tickless operation happens automatically on any core on which the Linux scheduler has only one thread to run. To check for this, look for that string in your Linux kernel config file. This file might be at */boot/<kernel version>* (determine your kernel version with "**uname -a**") or at */proc/config.gz*.

If the kernel was not built with CONFIG_NO_HZ_FULL, it might still be possible to run tickless by configuring it in the grub file (see the Grub File section). Specify the same set of CPUs for both **nohz_full** and **isolcpus**.

3.8 vt.handoff

vt.handoff (vt = virtualterminal) is a kernel boot parameter unique to Ubuntu, and is not an upstream kernel boot parameter. Its purpose is to allow the kernel to maintain the current contents of video memory on a virtual terminal. Therefore, when the operating system is booting up, when it moves past the boot loader, **vt.handoff** allows showing of an aubergine background, with Plymouth displaying a logo and progress indicator bar on top of this. Once the display manager comes up, it smoothly replaces this with a login prompt.

3.9 NUMA Balancing and MCE

NUMA Balancing inside a kernel automatically optimizes a task scanner for scheduling on the fly, This should be disabled to get a consistent performance during benchmarking.

Also, machine check event exceptions logging is disabled.

3.10 Active-State Power Management

Active-State Power Management (ASPM) saves power in the PCIe subsystem by setting a lower power state for PCIe links when the devices to which they connect are not in use. When ASPM is enabled, device latency increases because of the time required to transition the link between different power states. Therefore, ASPM support is disabled here for performance benchmarking.

Did this document help answer your questions?

3.11 High Performance of Small Packets on 100G NIC- Use 16 Bytes Rx Descriptor Size

ICE PMD supports both 16 and 32 bytes Rx descriptor sizes. The 16 bytes size can provide high performance at small packet sizes. Configuration of CONFIG_RTE_LIBRTE_ICE_16BYTE_RX_DESC in config files can be changed to use 16 bytes size Rx descriptors.

More details to set 16 bytes Rx descriptor size in Setting 16 Bytes Rx Descriptor Size.

3.12 Downloading and Installing the ice Driver

NOTE

This section can be skipped if only the DPDK driver is being used.

Intel® Ethernet 800 Series Linux Drivers for PF and VF are available from the following sources:

- http://sourceforge.net/projects/e1000/files/
- https://www.intel.com/content/www/us/en/download-center/home.html

Refer to the *Intel® Ethernet Controller E810 Feature Support Matrix* for the recommended driver combinations. For further feature explanation and configuration instructions, refer to the *ice* driver README.

1. Download and extract the *ice* driver.

   Run the following commands using sudo.

```
### download ice-<x.x.x>.tar.gz
tar -xzvf ice-<x.x.x.tar.gz
cd ice-<x.x.x>/src/
```

2. Compile and install the *ice* driver.

```
make -j 8
make install
modprobe ice
```

3. If an issue is encountered when loading the driver, use the following command to check dmesg for errors from the *ice* module.

```
dmesg | grep ice
```


Did this document help answer your questions?

3.13 Getting the Latest DPDK Code

NOTE

Always refer to the latest user support documentation provided on dpdk.org.

DPDK official releases and LTS versions are available for download from https://core.dpdk.org/download/.

NOTE

DPDK may also be cloned and downloaded using Git from git://dpdk.org/dpdk or http://git.dpdk.org/dpdk-stable/.

```
mkdir /usr/src/dpdk_latest/
cd /usr/src/dpdk_latest/

git clone git://dpdk.org/dpdk
##switch to releases branch to checkout version 22.11
git checkout releases
```

3.14 Prerequisite Library

Refer to dpdk.org for extended requirements details and optional tools. The following prerequisites include the base requirements for most setup scenarios.

Development Tools:

General development tools such as a supported C compiler are required.

To install:

- For Ubuntu:

```
apt install build-essential
```

- For RHEL:

```
dnf groupinstall "Development Tools"
```

NUMA:

NUMA is required by most modern machines, but not needed for non-NUMA architectures.

NOTE

For compiling the NUMA lib, run **libtool -version** to ensure that the **libtool** version is greater than or equal to 2.2. Otherwise, the compilation will fail with errors.

For Ubuntu:

```
sudo apt-get install libnuma-dev
```


Did this document help answer your questions?

For RHEL:

```
yum install numactl-devel
```

or:

```
git clone https://github.com/numactl/numactl.git
cd numactl
./autogen.sh
./configure
make install
```

The NUMA header files and lib file are generated in the *include* and *lib* folders, respectively, under <*numa install dir*>.

Python:

For Ubuntu:

```
apt-get install python3
```

Create a symlink to it: `sudo ln -s /usr/bin/python3 /usr/bin/python`

NOTE

A dependency has been added for building DPDK on Linux or FreeBSD: The Python module **pyelftools** (version 0.22 or greater), often packaged as python3-pyelftools, is required.

If not available as a distribution package, it can be installed with:

```
pip3 install pyelftools
```

or, if using Ubuntu, with:

```
apt install python3-elftools
```

or, download **pyelftools** from https://pypi.org/project/pyelftools/#files

```
tar -xzvf pyelftools-0.27.tar.gz
cd pyelftools-0.27
python setup.py install
pip3 install pyelftools
```

Meson and Ninja:

The Meson and Ninja tools are required to configure the DPDK build.

For Ubuntu:

```
sudo apt install meson ninja-build
```


Did this document help answer your questions?

For RHEL:

```
sudo dnf install meson ninja-build
```

If MESON is not available as a suitable package, it can also be installed using the Python 3 pip tool:

```
sudo apt install python3-pip
pip3 install meson
```

NOTE

pip3 puts the executable under */usr/local/lib/python3.8/dist-packages*, so put that directory in your *PATH*.

For example:

```
PATH="/usr/local/lib/python3.8/dist-packages:$PATH"
```

NOTE

If the following error is seen: "Requires >=0.47.1 but the version of Meson is 0.45.1"

First, remove meson installed by **apt** if installed:

```
sudo apt purge meson -y
```

Then, install via pip3:

```
sudo pip3 install meson
```

Then, make a symlink in bin folder from local bin:

```
sudo ln -s /usr/local/bin/meson /usr/bin/meson
```

Now, check:

```
which meson && meson --version
```


Did this document help answer your questions?

3.15 Installing DPDK

3.15.1 Installing DPDK Using the Meson Build System (Recommended)

1. Untar DPDK, if downloaded version is the dpdk-22.11.tar.xz tarball.

```
tar xJf dpdk-22.11.tar.xz
cd dpdk-22.11
```

2. Run the following set of commands from the top-level DPDK directory:

```
meson build
cd build
```

3. Run the following set of commands from the build directory:

```
ninja
sudo ninja install ldconfig
```

NOTE

The **meson configure** option could be used to enable Debug mode:

```
meson configure -Dbuildtype=debug
```

3.15.2 Setting 16 Bytes Rx Descriptor Size

For better small packet size performance, setting 16 bytes Rx descriptor is recommended. It can be set from the top-level DPDK directory:

For an l3fwd application, DPDK meson build settings could be used to set Rx descriptors to 16:

```
CC=gcc meson -Dlibdir=lib -Dexamples=l3fwd -Dc_args=-
DRTE_LIBRTE_ICE_16BYTE_RX_DESC --default-library=static x86_64-nativelinuxapp-gcc
```

3.16 Linux Drivers

Linux drivers handle PCI enumeration and link status interrupts in user mode, instead of being handled by kernel.

To work properly, different PMDs might require different kernel drivers. Depending on the PMD being used, a corresponding kernel driver should be loaded, and network ports should be bound to that driver.

- VFIO driver is a robust and secure driver that relies on IOMMU protection.
- UIO is a small kernel module to set up the device, map device memory to user space, and register interrupts.

Did this document help answer your questions?

Since DPDK release 1.7 onward provides VFIO support, it is recommended that *vfio-pci* be used as the kernel module for DPDK-bound ports in all cases. This is a more robust and secure driver compared to UIO, relying on IOMMU protection. To make use of VFIO, the *vfio-pci* module must be loaded.

If an IOMMU is unavailable, the *vfio-pci* can be used in no-iommu mode. If, for some reason, vfio is unavailable, the UIO-based modules, *igb_uio* and *uio_pci_generic* may be used.

NOTE

For UIO module installation instructions, see section 7.4 UIO on the dpdk.org Linux Drivers page (http://doc.dpdk.org/guides/linux_gsg/linux_drivers.html)

3.16.1 Loading the *vfio-pci* Module

Load the *vfio-pci* module with the following command:

```
modprobe vfio-pci
```

VFIO kernel is usually present by default in all distributions. If it is not, consult your distribution's documentation for installation instructions.

To make use of full VFIO functionality, both kernel and BIOS must support and be configured to use I/O virtualization (such as Intel® VT-d).

In most cases, specifying `iommu=on` as kernel parameter should be enough to configure the Linux kernel to use IOMMU.

NOTE

VFIO no-IOMMU mode:

If there is no IOMMU available on the system, VFIO can still be used, but it must be loaded with an additional module parameter:

```
modprobe vfio enable_unsafe_noiommu_mode=1
```

Alternatively, one can also enable this option in an already loaded kernel module:

```
echo 1 > /sys/module/vfio/parameters/enable_unsafe_noiommu_mode
```

After that, VFIO can be used with hardware devices as usual.

Note that it might be required to unload all VFIO-related modules before probing the module again with the `enable_unsafe_noiommu_mode=1` parameter.

Warning: Since no-IOMMU mode forgoes IOMMU protection, it is inherently unsafe. That said, it does make it possible for the user to keep the degree of device access and programming that VFIO has, in situations where IOMMU is not available.

Did this document help answer your questions?

3.16.2 Binding and Unbinding Network Ports

1. The *dpdk-devbind* utility may be accessed and run through */usr/local/bin* or through the usertools directory in the top-level DPDK directory:

```
/usr/local/bin/dpdk-devbind.py
```

or

```
./dpdk-stable-22.11.0/usertools/dpdk-devbind.py
```

To display available devices, run the following command:

```
./dpdk-devbind.py -s
```

or

```
./dpdk-devbind.py –status
```

2. If a device is in use, it will be listed as **Active**. In this state, the interface cannot be unbound. To change the state to inactive, bring down the interface. Then, the port is free to bind to the desired module.

```
ifdown ethX
```

or

```
ip link set dev ethX down
```

To make the interface available to DPDK, bind it to *vfio-pci*. In the following steps, `<B:D.F>` is the Bus Device Function (BDF) number of the ethX interface. The BDF number can be found using various Linux commands or the *get_config.sh*.

Unbind from previous module or driver:

```
./dpdk-devbind.py -u  <B:D.F>
```

Bind to *vfio-pci* for use by DPDK:

```
./dpdk-devbind.py -b vfio-pci <B:D.F>
```

Or bind to *ice* for use by kernel driver:

```
./dpdk-devbind.py -b ice <B:D.F>
```


Did this document help answer your questions?

4.0 Virtual Function (VF) Setup with DPDK

1. Confirm **IOMMU** and virtualization technologies are enabled in the BIOS and Linux kernel.
   a. In the BIOS, check that Intel® VT and Intel® VT-d are enabled. On certain servers SR-IOV may need to be enabled in the BIOS as well.
   b. Update the kernel boot parameters to enable IOMMU support. **IOMMU** enables mapping of virtual memory addresses to physical addresses. The following arguments turn on **iommu** and set it to pass-through mode:

   ```
   intel_iommu=on
   iommu=pt
   ```

   The parameters can be added by editing the */etc/default/grub* file, or by executing a grubby command, shown below:

   ```
   grubby --args="intel_iommu=on iommu=pt" --update-kernel DEFAULT
   ```

   c. Reboot the system for the change to take effect.
2. Install the kernel *iavf* driver for initial VF creation and optional configuration.

   ```
   tar xzvf iavf-x.x.x
   cd iavf-x.x.x/src/
   ```

3. Compile and install the *iavf* driver.

   ```
   make -j 8
   make install
   modprobe iavf
   ```

4. Create *n* number of VFs on the chosen interface.

   The following command shows 4 VFs are created on eth0.

   ```
   echo 4 > /sys/class/net/eth0/device/sriov_numvfs
   ```

5. Certain configuration can be done with the kernel *iavf* driver, such as adding MAC Addresses to the VFs or setting trust mode.

   For example, the following commands will first set VF 0 MAC to the shown MAC address and then enable trusted mode:

   ```
   ip link set eth0 vf 0 mac 68:05:ca:a6:0a:b1
   ip link set eth0 vf 0 trust on
   ```

6. Bind the VF interface to DPDK using the vfio-pci module, the same method as used for PF interfaces.

Did this document help answer your questions?

To do this, use the BDF number, which can be found using various Linux commands or the `get_config.sh`.

```
./dpdk-devbind.py -b vfio-pci <B:D.F>
```

7. Start the **testpmd** application on the VF.

   The `-a` EAL option can be used to explicitly point to the VF device. For example:

   ```
   sudo ./dpdk-testpmd -n 4 -a 18:01.0 -- --rxq=4 --txq=4 -i --forward-mode=mac
   ```

8. Debug logging with **testpmd** is available for the *iavf* driver.

   This can be enabled with the EAL flag `--log-level="pmd.net.iavf,debug”`. This will print out debug messages from PMD_INIT_LOG(DEBUG, “message”) statements in the *iavf* driver. For example:

   ```
   sudo ./dpdk-testpmd -n 4 -a 18:01.0 --log-level="pmd.net.iavf,debug" --
   --rxq=4 --txq=4 -i --forward-mode=mac
   ```


Did this document help answer your questions?

5.0 Advanced Features and Debug

5.1 Q-in-Q Support

Q-in-Q, also known as double virtual LAN (VLAN) or VLAN stacking, is a standardized, networking technique that allows a packet to be encapsulated by two or more VLAN tags. VLAN complies with the IEEE standard 802.1Q, and is a method of segmenting traffic by adding an additional L2 header to tag Ethernet frames with a given ID. Q-in-Q adds two headers (or VLAN tags) to an Ethernet packet instead of just one. Q-in-Q complies with the IEEE 802.1ad specification.

Q-in-Q and single VLAN provide similar benefits like added security through network segmentation and logical organization of large networks. While VLAN provides approximately 4K segmentations, the additional tag in Q-in-Q expands this range to over 16 million separate VLANs.

DPDK supports Q-in-Q on both PF and VF interfaces, used either in the host environment or in a VM. Because of this flexibility, proper Q-in-Q configuration varies depending on the environment and the desired outcome. The following commands and examples are provided as a starting point for exploring this feature.

- To enable Q-in-Q in an active testpmd session, use the following command:

```
testpmd> vlan set extend on (port_id)
```

- To set inner and outer TPID for packet filtering on a port, use the following command:

```
testpmd> vlan set (inner | outer) tpid (value) (port_id)
```

VLAN stripping is also available for both inner and outer tags.

- To strip the outer VLAN on Q-in-Q packets use the following command. This command will strip the VLAN tag on single VLAN packets as well.

```
testpmd> vlan set strip on (port_id)
```

- To strip the inner tag from Q-in-Q packets use the following command:

```
testpmd> vlan set qinq_strip on (port_id)
```


Did this document help answer your questions?

In the following example, VLAN and Q-in-Q handling are enabled on port 0, and inner and outer tag filters are configured for different EtherTypes. Stripping of the outer VLAN tag is enabled on the port as well.

```
testpmd> vlan set filter on 0
testpmd> vlan set extend on 0
testpmd> vlan set inner tpid 0x8100 0
testpmd> vlan set outer tpid 0x88A8 0
testpmd> vlan set strip on 0
```

For additional VLAN configuration options with testpmd, see the Testpmd Runtime Functions page in the dpdk.org documentation:

https://doc.dpdk.org/guides/testpmd_app_ug/testpmd_funcs.html

5.2 Malicious Driver Detection

Some Intel® Ethernet devices use Malicious Driver Detection (MDD) to detect malicious traffic from the VF, and disable Tx/Rx queues or drop the offending packet until a VF driver reset occurs.

The E810 offers various extended message levels that are enabled using ethtool. One such example is `tx_err`. This level allows for additional Tx MDD-related output. Similarly `rx_err` can be enabled for Rx MDD-related output.

Enabling and viewing this output is useful when using a DPDK VF with a Linux kernel PF, that is, an *iavf* DPDK PMD and a kernel *ice* driver. If using a DPDK PF driver (such as, an *ice* PMD), view the DPDK application logs for MDD event notifications.

Message levels are set using ethtool. To enable `tx_err` messaging, use the following command:

```
ethtool -s ens5f0 msglvl tx_err on
```

The output will print to dmesg. For example:

```
[ +9.188014] ice 0000:86:00.0: Malicious Driver Detection event 7 on TX queue 113
PF# 0 VF# 0
[ +0.000006] ice 0000:86:00.0: Malicious Driver Detection event TX_TCLAN detected
on PF
[ +0.000003] ice 0000:86:00.0: Malicious Driver Detection event TX_TCLAN detected
on VF 0
[ +0.000003] ice 0000:86:00.0: 1 Tx Malicious Driver Detection events detected on
PF 0 VF 0 MAC 3a:14:8e:da:b0:98.
```

Devlink health reporting can be used to view additional MDD information:

```
 devlink health dump show pci/0000:86:00.1 reporter mdd
```

A capability new to the E810 is the ability to enable and disable automatic VF resets upon MDD detection. Refer to the kernel driver ice README, section *Malicious Driver Detection (MDD) for VFs*, for more details on this feature.

Did this document help answer your questions?

5.3 Receive Side Scaling Configuration

The Receive Side Scaling (RSS) hashing algorithm relies on a combination of different attributes to calculate the hash value. These attributes include the source and destination IP addresses, protocol type, and port numbers associated with each incoming packet. By considering these factors, the algorithm distributes the network traffic evenly across multiple receive queues, facilitating efficient load balancing and maximizing the performance of the system. To ensure the accuracy of the RSS hash calculation and gain a deeper understanding of the process, the following command can be used:.

```
testpmd> set verbose 8
```

This command enables verbose mode in DPDK, providing more detailed output to observe the calculated hash values for each packet and understand how DPDK determines the appropriate receive queue for packet distribution.

For more details on RSS configuration and testing, refer to the DPDK.org test plan:

https://doc.dpdk.org/dts/test_plans/index.html

Did this document help answer your questions?

6.0 Test Applications (pktgen and testpmd)

The **pktgen** application is traffic generator powered by DPDK. It is capable of generating traffic at wire rate. For more information on the **pktgen** application, see:

https://pktgen-dpdk.readthedocs.io/en/latest/

Testpmd is another reference application distributed with the DPDK package. Its main purpose is to forward packets between Ethernet ports on a network interface. The **testpmd** application provides a number of different throughput tests and access NIC hardware features, such as Intel® Flow Director. For more information on the testpmd application, see:

https://doc.dpdk.org/guides/testpmd_app_ug/index.html

6.1 Setting Up pktgen Server

1. Get the latest **pktgen** application from git repository to the DPDK root directory.

   ```
   git clone git://dpdk.org/apps/pktgen-dpdk
   ```

   Or, download from:

   http://git.dpdk.org/apps/pktgen-dpdk

2. Download and install lua.

   Lua is used in **pktgen** to script and configure the application and also to plug into DPDK functions to expose configuration and statistics.

   For Ubuntu:

   ```
   sudo apt-get install liblua5.3-0 liblua5.3-dev libpcap-dev libbsd-dev
   ```

   For RHEL:

   Download *lua-5.4.4.tar.gz* from https://www.lua.org/download.html, then copy it to a peer system like */usr/local/src/* and extract it:

   ```
   # tar zxf lua-5.4.4.tar.gz
   # cd lua-5.4.4/
   # make linux install
   ```

   a. Download *libpcap-devel* from https://pkgs.org/download/libpcap-devel and install it:

      ```
      # rpm -ivh libpcap-devel*
      ```


Did this document help answer your questions?

3. Install **pktgen**.

   If git clone is used to download **pktgen**, use:

```
cd pktgen-dpdk
make
```

   The **pktgen** application is now under root */pktgen-dpdk/Builddir/app* directory:

```
root@user:/pktgen-dpdk/Builddir/app#./pktgen
```

6.2 Setting Up Testpmd Application

The testpmd application is built automatically when DPDK is installed using Meson, as described in Installing DPDK Using the Meson Build System (Recommended). Testpmd may be accessed and run through */usr/local/bin* or through the app directory in the DPDK build directory:

```
/usr/local/bin/dpdk-testpmd
```

or

```
./dpdk-stable-22.11.0/build/app/dpdk-testpmd
```

NOTE

On Fedora you will need to add */usr/local/lib64* to your ld path; it is not there by default.

Testpmd should be run with *sudo* or as the root user.

6.2.1 Running Testpmd

When running **testpmd** there are two distinct parts to the command-line options – the first half are the EAL parameters, followed by the **testpmd** command-line options as the second half. These sections are separated in the command with a `--` separator.

For example, a simple command to start **testpmd** looks like:

```
./dpdk-testpmd -a 00:01.0 -- -i
```

This command explicitly passes in the PCI device located at BDF number `00:01.0` with the `-a` flag in the EAL section of the command. Then the `-i` flag in the **testpmd** section starts the application in interactive mode.

For more details on the role of EAL, see the DPDK Overview.

Did this document help answer your questions?

Common Command-line Options

This section lists several common flags for EAL parameters and **testpmd** parameters. These commands are intended to provide a starting place when learning the application. Several of these flags have more details available on dpdk.org. See Example Testpmd Configuration for an extended example of running **testpmd**.

Some common flags for **EAL parameters** include:

- `-a, --allow <[domain:]bus:devid.func>`: Add a PCI device in to the list of devices to probe.
- `-n <number of channels>`: Set the number of memory channels to use.
- `-l <core list>`: List of cores to run on, where – is used as a range separator and , is used as a single number separator.
- `--socket-mem <amounts of memory per socket>`: Preallocate specified amounts of memory per socket. The parameter is a comma-separated list of values.

For a complete list of EAL parameters, see dpdk.org:

https://doc.dpdk.org/guides/linux_gsg/linux_eal_parameters.html

Some common flags for **testpmd options** include:

- `-i, --interactive`: Start **testpmd** with an interactive prompt. See Testpmd Runtime Functions for more details.
- `--rxq=N`: Set number of RX queues per port to N.
- `--txq=N`: Set number of TX queues per port to N.
- `--nb-cores=N`: Set number of forwarding cores.
- `--forward-mode=mode`: Set forwarding to a specific mode such as mac, rxonly, ieee1588, noisy, etc. The default mode is io.

For a complete list of **testpmd** command-line options, see dpdk.org:

https://doc.dpdk.org/guides/testpmd_app_ug/run_app.html

6.2.2 Testpmd Runtime Functions

When **testpmd** is started in interactive mode using the `-i` or `--interactive` option, a prompt is displayed that allows for real time configuration, statistic read-outs, and the ability to start/stop packet forwarding. Many of the **testpmd** command-line options can also be accessed as runtime functions.

An extensive number of **testpmd** runtime functions are available that are broken into multiple categories. See dpdk.org for a complete list:

https://doc.dpdk.org/guides/testpmd_app_ug/testpmd_funcs.html

Did this document help answer your questions?

To help with getting started, here are some examples of common runtime functions:

- `set verbose (level)`: Set the debug verbosity level.
- `set fwd (io|mac|macswap|flowgen|rxonly|txonly|csum|icmpecho|noisy|5tswap|shared-rxq) (""|retry)`: Set the packet forwarding mode.
- `show port (info|summary|stats|xstats|fdir|dcb_tc|cap) (port_id|all)`: Display information for a given port or all ports.
- `show fwd stats all`: View statistics for all ports that were collected beginning from the time the forwarding engine was started.
- `start`: Start packet forwarding with current configuration.
- `stop`: Stop packet forwarding and display accumulated statistics.

6.2.3 Debugging in Testpmd

Debug logging is available in **testpmd** and can be enabled by an EAL command-line parameter.

To enable ice PMD debug logging, add the `--log-level="pmd.net.ice,debug”` flag to the **testpmd** EAL parameters. This will print debug messages from PMD_INIT_LOG(DEBUG, “message”):

```
sudo ./dpdk-testpmd -a 00:01.0 --log-level=”pmd.net.ice,debug" -- -i
```

6.3 Example pktgen Configuration

This section provides a more complex example of a **pktgen** command.

This command starts pktgen with these EAL parameters:

- `-l <core list>`: List of cores to run on, where ‘-‘ is used as a range separator and ‘,’ is used as a single number separator.
- `-n <number of channels>`: Set the number of memory channels to use.
- `--proc-type`: Type of this process.
- `--log-level`:
- `--socket-mem`: Memory to allocate on specific sockets (use comma separated values)
- `--file-prefix`: Prefix for hugepage filenames
- `-a, --allow <[domain:]bus:devid.func>`: Add a PCI device into the list of devices to probe.

For a complete list of EAL parameters, see dpdk.org:

https://pktgen-dpdk.readthedocs.io/en/latest/usage_eal.html

For a complete list of command-line options, see dpdk.org:

https://pktgen-dpdk.readthedocs.io/en/latest/usage_pktgen.html

Did this document help answer your questions?

1. Example **pktgen** configuration command:

```
./pktgen -l 24-32 -n 4 --proc-type auto --log-level 7 --socket- mem=0,1024
--file- prefix pgb2000 -a b2:00.0 -- -N -P -T -m [25-28:29-32].0
```

2. **Pktgen** entropy configuration to enable multiple traffic streams with RSS:

```
Pktgen:/> set 0 proto tcp
Pktgen:/> set 0 size 128
Pktgen:/> set 0 src mac 68:05:ca:a6:0b:1c
Pktgen:/> set 0 dst mac 68:05:ca:a6:0a:b0
Pktgen:/> set 0 src ip 192.168.103.101/24
Pktgen:/> set 0 dst ip 192.168.103.102
Pktgen:/> enable 0 range
Pktgen:/> range 0 proto tcp
Pktgen:/> range 0 size 128 128 128 1
Pktgen:/> range 0 src mac 68:05:ca:a6:0b:1c 68:05:ca:a6:0b:1c
68:05:ca:a6:0b:1c 00:00:00:00:00:01
Pktgen:/> range 0 dst mac 68:05:ca:a6:0a:b0 68:05:ca:a6:0a:b0
68:05:ca:a6:0a:b0 00:00:00:00:00:01
Pktgen:/> range 0 src ip 192.168.103.101 192.168.103.101 192.168.103.101
0.0.0.1
Pktgen:/> range 0 dst ip 192.168.103.102 192.168.103.102 192.168.103.102
0.0.0.1
Pktgen:/> range 0 dst port 2000 2000 2000 1
Pktgen:/> range 0 src port 3000 3000 3016 1
Pktgen:/> start 0

Note: The above configuration can be saved to a file and loaded directly to
the pktgen.
Pktgen:/> save <path-to-file>
Pktgen:/> load <path-to-file>
```

To view the range config, use the following command:

```
Pktgen:/> page range
```

To switch back to packets view, use the following command:

```
Pktgen:/> page main
```

Also refer to:

https://pktgen-dpdk.readthedocs.io/en/latest/commands.html#runtime-options-and-command

Did this document help answer your questions?

6.4 Example Testpmd Configuration

This section provides a more complex example of a **testpmd** command.

This command starts testpmd with the following EAL parameters:

- `-l 2-11`: Run DPDK on cores 2 through 11.
- `-n 4`: Use 4 memory channels.
- `-a 11:00.2`: Probe PCI device at BDF number 11:00.2.
- `--file-prefix testpmd18000`: Run DPDK under the specified prefix 'testpmd18000'.
- `--socket-mem=1024`: Pre-allocates 1024 megabytes on socket 0.
- `--proc-type=auto`: Set the type of the process to auto.

And, the following testpmd parameters:

- `--nb-cores=4`: Set the number of cores used by the application for forwarding to 4.
- `--rxq=4`: Set the number of RX queues per port to 4.
- `--txq=4`: Set the number of TX queues per port to 4.
- `-i`: Start testpmd in interactive mode.
- `--forward-mode=mac`: Set the packet forwarding mode to mac, which changes the source and destination addresses of the packets before forwarding them.
- `--eth-peer=0,68:05:ca:c1:c9:29`: Set the MAC address of the peer port.

The following is the testpmd output from running the command:

```
./dpdk-testpmd -l 2-11 -n 4 -a 11:00.2 --file-prefix testpmd18000 --socket-
mem=1024,0 --proc-type=auto -- --nb-cores=4 --rxq=4 --txq=4 -i --forward-mode=mac
--eth-peer=0,68:05:ca:c1:c9:29

EAL: Detected 88 lcore(s)
EAL: Detected 2 NUMA nodes
EAL: Auto-detected process type: PRIMARY
EAL: Multi-process socket /var/run/dpdk/testpmd18000/mp_socket
EAL: Selected IOVA mode 'PA'
EAL: Probing VFIO support...
EAL: PCI device 0000:18:00.0 on NUMA socket 0
EAL:   probe driver: 8086:1592 net_ice
Interactive-mode selected
Set mac packet forwarding mode
testpmd: create a new mbuf pool <mbuf_pool_socket_0>: n=219456, size=2176,
socket=0
testpmd: preferred mempool ops selected: ring_mp_mc

Warning! port-topology=paired and odd forward ports number, the last port will
pair with itself.

Configuring Port 0 (socket 0)
Port 0: 68:05:CA:A6:0A:B0
Checking link statuses...
Done

testpmd> start
mac packet forwarding - ports=1 - cores=4 - streams=4 - NUMA support enabled, MP
allocation mode: native
Logical Core 3 (socket 0) forwards packets on 1 streams:
```


Did this document help answer your questions?

```
  RX P=0/Q=0 (socket 0) -> TX P=0/Q=0 (socket 0) peer=68:05:CA:A6:0B:1C
Logical Core 4 (socket 0) forwards packets on 1 streams:
  RX P=0/Q=1 (socket 0) -> TX P=0/Q=1 (socket 0) peer=68:05:CA:A6:0B:1C
Logical Core 5 (socket 0) forwards packets on 1 streams:
  RX P=0/Q=2 (socket 0) -> TX P=0/Q=2 (socket 0) peer=68:05:CA:A6:0B:1C
Logical Core 6 (socket 0) forwards packets on 1 streams:
  RX P=0/Q=3 (socket 0) -> TX P=0/Q=3 (socket 0) peer=68:05:CA:A6:0B:1C
mac packet forwarding packets/burst=32
nb forwarding cores=4 - nb forwarding ports=1
port 0: RX queue number: 4 Tx queue number: 4
Rx offloads=0x0 Tx offloads=0x10000
RX queue: 0
RX desc=1024 - RX free threshold=32
RX threshold registers: pthresh=8 hthresh=8  wthresh=0
RX Offloads=0x0
TX queue: 0
TX desc=1024 - TX free threshold=32
TX threshold registers: pthresh=32 hthresh=0  wthresh=0
TX offloads=0x10000 - TX RS bit threshold=32

testpmd> show port stats all

  ######################## NIC statistics for port 0  ########################
  RX-packets: 322932478810 RX-missed: 558861443  RX-bytes:  18446744341590657960
  RX-errors:  0
  RX-nombuf:  0
  TX-packets: 302564215177 TX-errors: 0          TX-bytes:  18446743108765253980

  Throughput (since last show)
  Rx-pps:     35183210
  Tx-pps:     32640093
  ############################################################################

testpmd> stop
Telling cores to stop...
Waiting for lcores to finish...

  ------- Forward Stats for RX Port= 0/Queue= 0 -> TX Port= 0/Queue= 0 -------
  RX-packets: 155985410964   TX-packets: 146417324571   TX-dropped: 9568086393

  ------- Forward Stats for RX Port= 0/Queue= 1 -> TX Port= 0/Queue= 1 -------
  RX-packets: 6224427006     TX-packets: 4972240065     TX-dropped: 1252186941

  ------- Forward Stats for RX Port= 0/Queue= 2 -> TX Port= 0/Queue= 2 -------
  RX-packets: 4981979331     TX-packets: 4973665526     TX-dropped: 8313805

  ------- Forward Stats for RX Port= 0/Queue= 3 -> TX Port= 0/Queue= 3 -------
  RX-packets: 155943833856   TX-packets: 146389491787   TX-dropped: 9554342069

  ---------------------- Forward statistics for port 0  ----------------------
  RX-packets: 323135651965   RX-dropped: 561082081     RX-total: 323696734046
  TX-packets: 302752721949   TX-dropped: 20382929208   TX-total: 323135651157
  ----------------------------------------------------------------------------

  ++++++++++++++ Accumulated forward statistics for all ports++++++++++++++
  RX-packets: 323135651965   RX-dropped: 561082081     RX-total: 323696734046
  TX-packets: 302752721949   TX-dropped: 20382929208   TX-total: 323135651157
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Done.
```


Did this document help answer your questions?

6.5 Running a Sample Application L3fwd Server

Following is an overview of the Layer 3 forwarding sample application, in which the forwarding decision is based on information read from the input packet. This is intended as a quick-start resource. For more information, see the *Sample Applications User Guides* at:

https://doc.dpdk.org/guides/sample_app_ug/l3_forward.html

1. To compile a sample application with Meson, use the configure command.

   A single application can be compiled by passing in the application's name. Alternatively, all sample applications can be compiled by specifying 'all'. Run the meson command from the DPDK 'build' directory.

```
### Move into build directory
cd dpdk-22.11/build

### Compile and build L3 Forwarding Sample Application
meson configure -Dexamples=l3fwd
ninja
```

2. Run the application from the examples directory under build.

   The full path would look like:

```
./dpdk-22.11/build/examples/dpdk-l3fwd
```

The following are some usage examples of the application.

4 cores command line option:

```
./dpdk-l3fwd -l 22,24,26,28 -n 6 -- -p 0x0 --config="(0,0,22),(0,1,24),(0,2,26),
(0,3,28)"
```

2 cores command line option:

```
./dpdk-l3fwd/build/l3fwd -l 3-4 -n 4 -a 18:00.0 -- -p 0x1 --parse-ptype --
config="(0,0,3),(0,1,4)" -P
```

In these commands:

- The **-l** option enables cores 22,24,26,28.
- The **-p** option enables port 0,
- The **-config** option enables one queue on each port and maps each (port,queue) pair to a specific core. For example, (0,0,22) maps queue 0 from port 0 to lcore 22.

Please review Section 20.3, Running the Application in the *Sample Applications User Guides* for a complete explanation of all flag options.

Example **l3fwd** output:

```
./dpdk-l3fwd -l 22,24,26,28 -n 6 -- -p 0x0 --config="(0,0,22),(0,1,24),(0,2,26),
(0,3,28)"
EAL: Detected 88 lcore(s)
EAL: Detected 2 NUMA nodes
```


Did this document help answer your questions?

```
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'PA'
EAL: Probing VFIO support...
EAL: PCI device 0000:00:04.0 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.1 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.2 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.3 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.4 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.5 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.6 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.7 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:3d:00.0 on NUMA socket 0
EAL:   probe driver: 8086:37d2 net_i40e
EAL: PCI device 0000:3d:00.1 on NUMA socket 0
EAL:   probe driver: 8086:37d2 net_i40e
EAL: PCI device 0000:80:04.0 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.1 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.2 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.3 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.4 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.5 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.6 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.7 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:86:00.0 on NUMA socket 1
EAL:   probe driver: 8086:10c9 net_e1000_igb
EAL: PCI device 0000:86:00.1 on NUMA socket 1
EAL:   probe driver: 8086:10c9 net_e1000_igb
EAL: PCI device 0000:af:00.0 on NUMA socket 1
EAL:   probe driver: 8086:1592 net_ice
ice_load_pkg_type(): Active package is: 1.3.28.0, ICE OS Default Package
ice_init_proto_xtr(): Protocol extraction is not supported
LPM or EM none selected, default LPM on
Initializing port 0 ... Creating queues: nb_rxq=4 nb_txq=4... Port 0 modified RSS
hash function based on hardware support,requested:0xa38c configured:0x2288
 Address:68:05:CA:5C:CF:A8, Destination:02:00:00:00:00:00, Allocated mbuf pool on
socket 1
LPM: Adding route 192.18.0.0 / 24 (0)
LPM: Adding route 192.18.1.0 / 24 (1)
LPM: Adding route 192.18.2.0 / 24 (2)
LPM: Adding route 192.18.3.0 / 24 (3)
LPM: Adding route 192.18.4.0 / 24 (4)
LPM: Adding route 192.18.5.0 / 24 (5)
LPM: Adding route 192.18.6.0 / 24 (6)
LPM: Adding route 192.18.7.0 / 24 (7)
LPM: Adding route 2001:200:: / 48 (0)
LPM: Adding route 2001:200:0:0:1:: / 48 (1)
LPM: Adding route 2001:200:0:0:2:: / 48 (2)
LPM: Adding route 2001:200:0:0:3:: / 48 (3)
LPM: Adding route 2001:200:0:0:4:: / 48 (4)
LPM: Adding route 2001:200:0:0:5:: / 48 (5)
LPM: Adding route 2001:200:0:0:6:: / 48 (6)
LPM: Adding route 2001:200:0:0:7:: / 48 (7)
txq=22,0,1 txq=24,1,1 txq=26,2,1 txq=28,3,1

Initializing rx queues on lcore 22 ... rxq=0,0,1
```


Did this document help answer your questions?

```
Initializing rx queues on lcore 24 ... rxq=0,1,1
Initializing rx queues on lcore 26 ... rxq=0,2,1
Initializing rx queues on lcore 28 ... rxq=0,3,1

Checking link
status.....................................................................
..............done
Port 0 Link Down
L3FWD: entering main loop on lcore 24
L3FWD:  -- lcoreid=24 portid=0 rxqueueid=1
L3FWD: entering main loop on lcore 26
L3FWD:  -- lcoreid=26 portid=0 rxqueueid=2
L3FWD: entering main loop on lcore 22
L3FWD:  -- lcoreid=22 portid=0 rxqueueid=0
L3FWD: entering main loop on lcore 28
L3FWD:  -- lcoreid=28 portid=0 rxqueueid=3
```

6.6 DPDK Test Plans

For further use case and configuration examples, see the Test Plans section of dpdk.org:

https://doc.dpdk.org/dts/test_plans/index.html

Did this document help answer your questions?