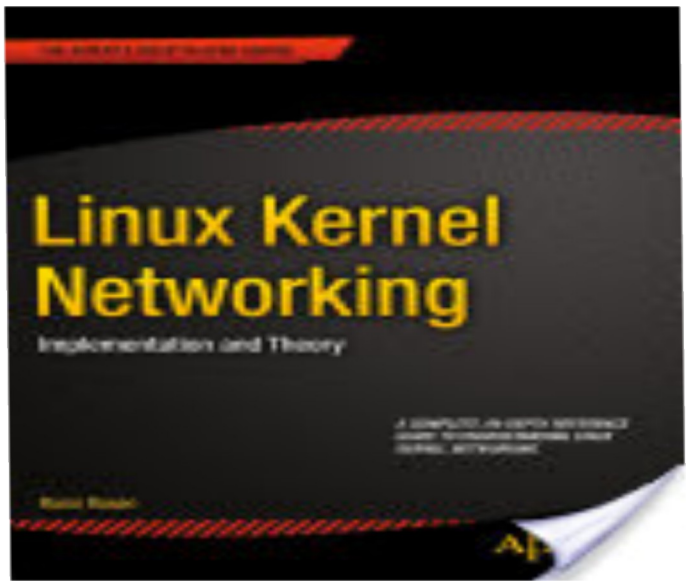
DPDK in depth

Rami Rosen

Kernel TLV
August 2018

About myself

- Rami Rosen, author of “Linux Kernel Networking”, Apress; Linux Kernel Expert. **Website**: http://ramirose.wixsite.com/ramirosen



- **Chinese translation of the book** →

Agenda

- DPDK background and short history
- DPDK projects
- DPDK libraries and PMDs
- DPDK advantages and disadvantages
- DPDK Development model
- Anatomy of a simple DPDK application (*l2fwd*)
- Testpmd: DPDK CLI tool

DPDK Background

- **DPDK** (Data Plane Development Kit) is a User Space Open Source project, deals with IO acceleration, primarily for Data Centers.
  - For Linux and BSD.
  - Some work is done for Windows.
- **2010**: started by **Intel**.
- **2013**: ownership moved to **6WIND**, who also started the dpdk.org site.
- 6WIND contributed many features to DPDK (like rte_flow). 6WIND were maintainers of MLX4/MLX5 till recently.
- **2017 (April):** the project moved to the Linux Foundation.
- Network acceleration has always been a subject which attracted the attention of network vendors and software developers/architects/researchers.
- Other projects in this arena:
  - **ODP** – OpenDataPlane (Linaro): https://www.opendataplane.org/
  - Focused primarily on ARM
  - Snabb (Lua): https://github.com/snabbco/snabb

DPDK Background (contd)

- Based on using hugepages (2M or 1GB) for boosting performance
- This reduces significantly TLB flushes.
- Numa Awareness
  - Every PCI device has a Numa Node associated with it.
    - /sys/bus/pci/devices/0000:04:00.0/numa_node

- Performance reports for recent releases (ranging from 16.11 to 18.05) for Mellanox and Intel NICs are available on: http://static.dpdk.org/doc/perf/
- These performance results are updated every new DPDK release.
- Tests with L3FWD app with IPv4. Full details about the setup.

DPDK Projects

- DPDK is used in a variety of Open Source projects. Following is a very partial list:
- VPP (FD.io project): https://wiki.fd.io/view/VPP
- Contrail vRouter (Juniper Network)
  - Open Source SDN controller.
- Sample VNFs (OPNFV)
  - Using librte_pipeline.
  - vFW (Virtual Firewalls)
  - vCGNAPT (NAT)
  - More (there are 5 VNFs in total as of now)
- DPPD-PROX (monitoring DPDK stats)

DPDK Projects (contd)

- *TRex* – stateful traffic generator, based on DPDK (FD.io project)
- https://wiki.fd.io/view/TRex
  - Developed by Hanoch Haim from Cisco.

- ***Collectd*** - System statistics collection daemon
- DPDK stats and DPDK events plugins were added to collectd.
  - https://collectd.org/

- Integrated with OpenStack and OPNFV solutions.

- *SPDK=Storage Performance Development Kit*
  - *https://github.com/spdk/spdk*

- More – plenty of results when searching in google.

DPDK Projects (contd)

- DTS
- DPDK Test Suit
- http://dpdk.org/git/tools/dts
- Written in Python
- An Open Source project
- Consists of over 105 functional tests and benchmarking tests.
- Works with IXIA (HW packet generator) and dpdk-pktgen (SW packet generator)
- Work is being done for adding support for IXIA Networks and TRex
- TRex is an Open Source DPDK packet generator hosted on FD.io
- DTS currently supports Intel, Mellanox and Cavium nics.
  - In settings.py you can find the Vendor ID/Device ID of the devices supported by DTS.
  - http://git.dpdk.org/tools/dts/tree/framework/settings.py
- Note: Apart from it, the DPDK project itself contains over 100 **unit tests** (written in “C”) as part of the DPDK tree, under the “test” folder.

DPDK Libraries and PMDs

- What is DPDK ? DPDK is not a network stack.
- You can divide the DPDK project development into four categories:
- Libraries
  - There are over 45 libraries
  - ***Core Libraries:***
    librte_eal, librte_mbuf, more.
  - *librte_ethdev (formerly called librte_ether)*
    - Implements network devices and their callbacks.
  - *librte_hash*
    - *Provides an API for creating hash tables for fast lookup*

DPDK Libraries and PMDs - contd

- PMDs (Poll Mode Drivers)
- **Ethernet network PMD drivers**
- There are over 20 PMD network drivers (under drivers/net (1Gb, 10Gb, 25 Gb, 40 Gb and 100Gb.)
- Some of the drivers have ***"base"*** subfolder, for code which is shared with kernel module.
- For example, ENA (Amazon), SFC (Solarflare Communications), Intel IXGBE, Intel I40E, Intel FM10K, and more).
- Mellanox mlx4/mlx5 PMDs use a bifurcated model.
- This means that they work in conjunction with their kernel driver.
- Most network Ethernet PMDs use uio mapping (by setting the RTE_PCI_DRV_NEED_MAPPING flag in the *drv_flags* of the *rte_pci_driver object)*
  - *Exceptions: mlx4, mlx5, mvpp2, netsvc, szedata, dpaa/dpaa2, ifc*
- **Virtual devices** - vdevs (PF_PACKET, TAP, more)
- **Crypto devices**
- **Eventdev devices**
- **Raw Devices (NXP)**

Network PMDs

- Each network PMD typically defines an *rte_pci_driver* object and sets its probe, remove and PCI ID table.
- It calls RTE_PMD_REGISTER_PCI() to register it to the system
  - This adds it to a global linked list, before DPDK app main() starts, using _attribute__(constructor)
- Creates an instance of the network object (*rte_eth_dev)* in its *probe()* callback and defines its RX callback and TX callback.
- With **Linux kernel network drivers**, it is enough to insmod the driver, and its RX callback will receive traffic.
- **With DPDK PMDs**, there is no such thing. Building the PMD creates a static library by default (you can also change it to be an .so)
- A **DPDK application** must be built and linked against that PMD library and call these RX and TX callbacks to process the traffic.

- Apart from it, each network PMD defines a set of callbacks, for handling various tasks, like setting MTU, setting MAC address, enabling promiscuous mode, etc.
- This is done by defining an *eth_dev_ops* object and its callbacks.
  - There are over **85** callbacks in the *eth_dev_ops* object.
  - It is parallel to the *net_device*_ops of the Linux kernel networking stack.

DPDK – Advantages and Disadvantages

- **Advantages:**
  - very good performance in L2 layer.
  - Upstreaming is easier comparing to the Linux kernel.
- **Disadvantages:**
  - no L3/L4 network stack.

- Solutions:
- **VPP** – a project originated from Cisco, started in 2002.
- Became an Open Source project under FD.io (Linux Foundation)
- Every DPDK PMD can be used (according to VPP mailing list)

- **TLDK** – L4 sockets (UDP, TCP, more).
  - Does not use the regular Berkeley SOCKET API

DPDK – Advantages and Disadvantages (contd)

- **KNI**
  - A Linux kernel module, part of the DPDK repo
  - Does not support 32 bit
  - From c*onfig/defconfig_i686-native-linuxapp-gcc*

```
# KNI is not supported on 32-bit
CONFIG_RTE_LIBRTE_KNI=n
```

  - Not efficient
  - Not in kernel mainline. Also candidate for deprecation from DPDK.
  - There were discussions over the dpdk-dev mailing list about an alternative solution called **KCP**, Kernel Control Path; There were 10 iterations of KCP patchset about half a year ago (TBD: date), but the status currently is that KCP is **paused**.
- OvS-DPDK

DPDK applications and tools

- Sample applications
- There are over 50 sample applications under the “examples” folder.
- These applications are documented in detail in the “DPDK Sample Applications User Guides” (255 pages for DPDK 18.05).
- Starting from a basic helloworld application, up to more complex applications (like l2fwd, l3fwd, and more).

- Tools/Utils
  - The most helpful is *testpmd*
- Will be discussed later.

- You can use *dpdk-procinfo* to get stats/extended stats
- *dpdk-procinfo runs as a secondary process.*
  - *./dpdk-procinfo -- -p 1 --stats*
  - *./dpdk-procinfo -- -p 1 --xstats*

DPDK – development model

- Each 3 months there is a new release of DPDK
- The releases are announced over the **dpdk-announce** mailing list (announce@dpdk.org)
- Usually, there are up to 5 or 7 Release Candidates (RCs) before each final release.
- The naming scheme is adopted from Ubuntu: yy:mm since April 2016 (DPDK 16.04)
- For example, in 2018 there are the following 4 releases (18.11 will be released in November):
- **18.02** – 1315 patches
- **18.05** - 1716 patches (Venky)
- **18.08** - **898** patches. **1,339,507** lines of code (only C files and headers).
- **18.11** (LTS release)
- Apart from it, there are LTS (Long Term Stable) releases, one per year.
- With support for 2 years.
- There is a strict deprecation process
- Deprecation notice should be sent over the mailing list a time ahead.

- New features are sometime marked as “rte_experimental” and can be removed without prior notice.

DPDK – development model (contd)

- Development is done by git patches over a public mailing list, dpdk-dev.
- Governance:
- **Technical Board** of 8 members
- The rule is that no more than 40% of the members can be of the same company
- Need 2/3 to remove a member.
- Meetings are open and held over IRC once in two weeks
- Minutes are posted over the dpdk-dev mailing list
- Discussions are about technical topics like adding a library, new features, and deciding if there is a dispute about a patch set.
- Minutes: https://core.dpdk.org/techboard/minutes/
- **Governance Board**
  - Budgets, legal, conferences.

L2FWD – a simple DPDK application

```
int main(int argc, char **argv)
{
   ret = rte_eal_init(argc, argv); /*parse EAL arguments */
   …
   ret = l2fwd_parse_args(argc, argv); /* parse application-specific arguments */
   ...
   /* Initialise each port */
   RTE_ETH_FOREACH_DEV(portid) {
    …
    ret = rte_eth_dev_configure(portid, 1, 1, &local_port_conf);
    …
    ret = rte_eth_dev_start(portid);
```

L2FWD – a simple DPDK application (contd)

```
struct rte_mbuf *m;
   /* ... */
   while (!force_quit) {
      /* ... */
      nb_rx = rte_eth_rx_burst((uint8_t) portid, 0, pkts_burst, MAX_PKT_BURST);
      port_statistics[portid].rx += nb_rx;
      for (j = 0; j < nb_rx; j++) {
          m = pkts_burst[j];
      /* ... */
      l2fwd_simple_forward(m, portid);
 }
```

- There are also several L3FWD samples under the “examples” folder, which has somewhat similar logic.
- In L3FWD, there is a static lookup table for IPv4 and IPv6.
- You can select either LPM (the default) or Exact Match.
- The lookup is done according to the destination IP address in the IP header.

You need perform a setup before running any DPDK app:
Setting number of hugepages.

- For example:

  echo256 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages

- Binding the NIC to DPDK is done by using *dpdk-devbind.py* script
  - For example, *dpdk-devbind.py -b uio_pci_generic 00:04.0*
  - *This will call the remove() callback of the kernel module associated with this PCI ID, if it is loaded.*
  - *The remove callback of the KMOD does not cause it to be unloaded.*
  - *When you done with running DPDK application, you can reload the kernel module associated with this PCI ID; for example, if the KMOD is ixgbe, this can be done by:*
  - *dpdk-devbind.py -b ixgbe 00:04.0*

    This will call the *probe()* method of the IXGBE kernel driver

- For binding, you can use either of the following three kernel modules:
- *uio_pci_generic* (a generic kernel module)
- *vfio-pci* (a generic kernel module)
  - Sometimes *vfio-pci* is needed when UEFI secure boot is enabled.
  - See: https://doc.dpdk.org/guides/linux_gsg/linux_drivers.html#vfio
  - vfio-pci module doesn't support the creation of virtual functions.
- *igb_uio*
  - A DPDK kernel module, not in mainline)
  - The igb_uio kernel module adds an entry called max_vfs in PCI sysfs.
    - Writing to this entry creates DPDK VFs.
    - *See dpdk/kernel/linux/igb_uio/igb_uio.c*

testpmd

- *Testpmd is an application like any other DPDK application.*
- *testpmd* provides a CLI which enables you various operations:
- Gather information about a port.
- Attach/Detach port in runtime.
- Using the rte_eth_dev_attach()/rte_eth_dev_detach() API.
- (Eventually invoking the rte_eal_hotplug_add()/ rte_eal_hotplug_remove())
- When detaching a port, we also call rte_eth_dev_release_port() to set the state of the device to be RTE_ETH_DEV_UNUSED.

- Send packets.
- Sniff, dump and parse the contents of packets.
- This is enabled when starting testpmd with *--forward-mode=rxonly*
- load DDP profile
- DDP is Dynamic Device Personalization
- Device programmability
  - https://software.intel.com/en-us/articles/dynamic-device-personalization-for-intel-ethernet-700-series

testpmd - contd

- load BPF
- developed by Konstantin Ananyev
  - ***bpf-load command from testpmd CLI.***
  - *Uses librte_bpf API*

DPDK application - contd

- All DPDK applications usually have two sets of parameters, separated by “--”
- The **first** set is the **EAL (Environment Abstraction Layer) parameters**, and are passed to the *rte_eal_init()* method.
  - For example, --log-level=8.
  - Another example: the legacy mode is enabled by specifying ***--legacy-mem*** in the EAL command line parameter
- The **second** set is the application-specific parameters.
- There are two modes in which DPDK memory subsystem can operate: dynamic mode, and legacy mode.

- The two most important data structures for understanding DPDK networking are *rth_ethdev* and *rte_mbuf*.
- **rte_ethdev** represents a network device, and is somewhat parallel to the Linux kernel net_device object.
- Every **rte_ethdev** should be associated with a ***bus*** (rte_bus object)
- ***rte_bus*** was Introduced in DPDK 17.02
- For many PMDs it is the PCI bus.
- Creating rte_eth_dev is done by:
  - *rte_eth_dev_allocate(const char *name)*

- **rte_mbuf** represents a network buffer and is somewhat parallel to the Linux kernel *sk_buff* object.
- Allocation of rte_mbuf is done by *rte_pktmbuf_alloc(struct rte_mempool *mp)*
- rte_mbuf object can be chained (multi segmented)
- This is implemented by the next pointer of *rte_mbuf* and *nb_segs*

DPDK Roadmap https://core.dpdk.org/roadmap/

- Next release, 18.11, is an LTS, so effort will be one for stabilizing and bug fixing.

new device specification (devargs) syntax
power management: traffic pattern aware power control
add MPLS to rte_flow encapsulation API
add metadata matching in rte_flow API
mlx5: add BlueField representors
mlx5: support VXLAN and MPLS encapsulations
failure handler for PCIE hardware hotplug
virtual device hotplug
tap and failsafe support in multi-process
SoftNIC support for NAT
eventdev ordered and atomic queues for DPAA2
libedit integration
noisy VNF forward mode in testpmd

XDP and DPDK

- XDP and DPDK
- David Miller netdev conference – talks against DPDK in context of XDP.
  - Qi Zhang patchset.
    - http://mails.dpdk.org/archives/dev/2018-August/109791.html
      V3 of the patchset was posted to dpdk-dev in August 2018
    - Seems a promising and a very interesting new DPDK direction.
  - Based on AF_XDP, a patchset by Björn Töpel, which was merged in Kernel 4.18.
    - https://lwn.net/Articles/750845/

- Device querying patchset
- added a struct called *rte_class*
- lib/librte_eal/common/include/rte_class.h

Links

- DPDK website: https://www.dpdk.org/
- DPDK API: http://doc.dpdk.org/api/
- DPDK Summit: https://dpdksummit.com/

- "Network acceleration with DPDK"
  - https://lwn.net/Articles/725254/

- "Userspace Networking with DPDK"
  - https://www.linuxjournal.com/content/userspace-networking-dpdk

testpmd- Device querying

- Device querying (will be included in 18.08, only 12 out of 25 patches in a patchset posted by Gaetan Rivet of 6WIND were applied till now)
- *show device bus=pci*
- tespmd> show device bus=pci
- 0x0x2d1b920: 0000:05:00.0:net_i40e
-
- tespmd> show device bus=vdev
- 0x0x2d074f0: eth_af_packet0:net_af_packet
  -
- testpmd> show device bus=vdev,driver=net_af_packet/class=eth
- 0x0x2d074f0: eth_af_packet0:net_af_packet

- [System requirements:
  - Note: DPDK cannot run on any kernel. There are minimum kernel, specified in the “System Requirements” section of the “Getting Started Guide for Linux” doc: , http://doc.dpdk.org/guides/linux_gsg/sys_reqs.html
  - As of now, Kernel version should be >= 3.2