DPDK - Agility, Flexibility, Elasticity

M Jay

Got DPDK Cookbook?
Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. No computer system can be absolutely secure. Learn more at intel.com, or from the OEM or retailer.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performance.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. 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 known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, Intel Atom and Xeon 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.

© 2018 Intel Corporation.
Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2®, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
Send your input to

M Jay

Muthurajan.Jayakumar@intel.com
All Information Here is Public Open Software

Refer [www.dpdk.org](http://www.dpdk.org) for source code and documentation.
• A Quick Overview
• Features Got Introduced in Recent Release, Nov. 2017
• We Listened To You – Making it Easy to Use
• Your Hands-On
• Call For Action
From a CPU perspective:
- Last-level-cache (L3) hit ~40 cycles
- L3 miss, memory read is ~70ns (140 cycles at 2GHz)
- Harder to address at 100Gb rates

The Problem Statement

<table>
<thead>
<tr>
<th>Packet Size</th>
<th>64 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>40G packets/second</td>
<td>59.5 million each way</td>
</tr>
<tr>
<td>Packet arrival interval</td>
<td>16.8 ns</td>
</tr>
<tr>
<td>2 GHz clock cycles/packet</td>
<td>33 cycles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packet Size</th>
<th>1024 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>40G packets/second</td>
<td>4.8 million each way</td>
</tr>
<tr>
<td>Packet arrival interval</td>
<td>208.8 ns</td>
</tr>
<tr>
<td>2 GHz clock cycles/packet</td>
<td>417 cycles</td>
</tr>
<tr>
<td>Eliminating</td>
<td>How?</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Interrupt Context Switch Overhead</td>
<td>Polling</td>
</tr>
<tr>
<td>Kernel User Overhead</td>
<td>User Mode Driver</td>
</tr>
<tr>
<td>Core To Thread Scheduling Overhead</td>
<td>Pthread Affinity</td>
</tr>
</tbody>
</table>
Both Rx Loop and Tx Loop – Do they get executed on same core or on different cores?

Why?

Why not?
How Many of You Used Run to Completion? How Many Pipelines?

Run-to-Completion model

Pipeline model

Rx → Work On Packet → Tx

Rx → Worker Thread 1 → Tx
→ Worker Thread 2 → Tx
→ Worker Thread n → Tx

DPDK Flexible and Scalable With # of Cores

DPDK – Data Plane Development Kit
What is the Usage Model – For All Sample Apps?

DPDK User Mode Application

Core 1
Object Cache for Core 1

Core 2
Object Cache for Core 2

Core 3
Object Cache for Core 3

Core 4
Object Cache for Core 4

Huge Page Memory

Port 0

Port 1

Port 2

Port 3
DPDK Framework - Fundamentals

DPDK Fundamentals

- Implements run-to-completion and pipeline models
- No scheduler - all devices accessed by polling
- Supports 32-bit and 64-bit OSs, with and without NUMA
- Scales across Intel Processors – low cost to high performance CPUs.
- Number of cores and processors is not limited
- Optimal packet allocation across DRAM channels
- Use of 2M and 1G hugepages and cache-aligned structures
- Uses bulk concepts - processing 'n' packets simultaneously
- Open source and BSD licensed

Network Functions (Cloud, Enterprise, Telco)

- Core libraries
  - Core functions such as memory management, software rings, timers, etc.
- Packet classification
  - Software libraries for hash/exact match, LPM, ACL, etc.
- Accelerated SW libraries
  - Common functions such as IP fragmentation, reassembly, reordering, etc.
- Stats
  - Libraries for collecting and reporting statistics
- QoS
  - Libraries for QoS scheduling and metering/policing
- Packet Framework
  - Libraries for creating complex pipelines in software.

- ETHDEV
  - PMDs for physical and virtual Ethernet devices
- CRYPTODEV
  - PMDs for HW and SW crypto accelerators
- EVENTDEV
  - Event-driven PMDs (HW & SW)
- SECURITY
  - Hardware-acceleration APIs

Copyright © 2018, Intel Corporation, All rights reserved.

*Other names and brands may be claimed as the
Network Functions (Cloud, Enterprise, Telco)

Core libraries:
- EAL
- MBuf
- Mempool
- Ring
- Timer

Packet classification:
- ACL
- HASH
- EFD
- LPM
- Membership
- Flow Classf.

Accelerated SW libraries:
- Distributor
- IP Frag
- Power
- GRO
- Pkt Dump
- Reorder
- GSO

Stats:
- Bit Rate
- Latency
- Job Stats
- Metrics
- Sched
- Meter

QoS:
- Pipeline
- Port
- Table

Packet Framework:
- ENIC
- MLX4
- ENCAP
- Ring
- VIRTIO
- DPAA

PMDs for physical and virtual Ethernet devices:
- AF Packet
- CXGBE
- I40E
- NFP
- TAP
- ThunderX
- E1000
- KNI
- PCAP
- VHOST
- VXLAN

PMDs for HW and SW crypto accelerators:
- AESNI GCM
- NULL
- ZUC
- AESNI MB
- OpenSSL
- DPAA
- ARMv8
- QAT
- MRVL
- DPAA2
- Scheduler
- KASUMI
- SNOW 3G

Hardware-acceleration APIs:
- DPAA
- Skeleton
- OcteonTX
- SW

Event-driven PMDs (HW & SW):
- Event
- DR
- TM

Accelerated SW libraries:
- Acceleration
- SW

Userspace:
- Core libraries

Kernel:
- OPTIMIZATION
- NOTICE

DPDK – Data Plane Development Kit

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others
Get the DPDK Cookbook

By Muthurajan Jayakumar (Intel), published on March 7, 2018

Download

About the Data Plane Development Kit (DPDK)

The Data Plane Development Kit (DPDK) is a set of libraries and drivers that significantly improves packet processing performance and throughput, allowing more time for developing data plane applications. Developers of virtual network functions can use DPDK to consolidate workloads into a single architecture, for a scalable and simplified packet processing solution.

Got DPDK Cookbook?
Agenda

A Quick Overview
Features Got Introduced in Recent Release, Nov. 2017
We Listened To You – Making it Easy to Use
Your Hands-On
Call For Action
Three New Sample Applications

How to See Source and Documentation Side by Side

1) Service Cores Sample App


2) Flow Classify Sample App


3) Flow RTE Flow Filtering Sample App

Libraries and APIs – Browsing The Source

**Software Eventdev Poll Mode Driver**


http://dpdk.readthedocs.io/en/v17.11/_sources/eventdevs/sw.rst.txt

**APIs**

http://dpdk.org/doc/api/

**All Examples**

http://dpdk.org/doc/api/examples.html
Browsing Driver .h files

**SoftNIC Driver header file**

http://dpdk.org/doc/api/rte_eth_softnic_8h_source.html
How To Browse Other Upcoming Projects

http://dpdk.org/browse/draft/

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>draft</td>
<td></td>
</tr>
<tr>
<td>dpdk-draft-cli</td>
<td>CLI library as librte.Cmdline replacement</td>
</tr>
<tr>
<td>dpdk-next-build</td>
<td>Preparation of a new build system</td>
</tr>
</tbody>
</table>
# How To Browse All of Them

http://dpdk.org/browse/

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpdk</td>
<td>Data Plane Development Kit</td>
</tr>
<tr>
<td>dpdk-stable</td>
<td>Data Plane Development Kit - stable releases</td>
</tr>
<tr>
<td>apps</td>
<td></td>
</tr>
<tr>
<td>pktgen-dpdk</td>
<td>Traffic generator powered by DPDK</td>
</tr>
<tr>
<td>spp</td>
<td>Soft Patch Panel - DPDK resources management framework</td>
</tr>
<tr>
<td>draft</td>
<td></td>
</tr>
<tr>
<td>dpdk-draft-cli</td>
<td>CLI library as librte_cmdsplus replacement</td>
</tr>
<tr>
<td>dpdk-next-build</td>
<td>Preparation of a new build system</td>
</tr>
<tr>
<td>next</td>
<td></td>
</tr>
<tr>
<td>dpdk-next-crypto</td>
<td>Preparation of pull requests for crypto drivers</td>
</tr>
<tr>
<td>dpdk-next-eventdev</td>
<td>Preparation of pull requests for eventdev</td>
</tr>
<tr>
<td>dpdk-next-net</td>
<td>Preparation of pull requests for networking drivers</td>
</tr>
<tr>
<td>dpdk-next-net-intel</td>
<td>Preparation of pull requests for Intel networking drivers</td>
</tr>
<tr>
<td>dpdk-next-net-mix</td>
<td>Preparation of pull requests for Mellanox networking drivers</td>
</tr>
<tr>
<td>dpdk-next-pipeline</td>
<td>Preparation of pull requests for packet framework</td>
</tr>
<tr>
<td>dpdk-next-qos</td>
<td>Preparation of pull requests for QoS traffic management</td>
</tr>
<tr>
<td>dpdk-next-virtio</td>
<td>Preparation of pull requests for virtio/switch</td>
</tr>
<tr>
<td>old</td>
<td></td>
</tr>
<tr>
<td>memnic</td>
<td>DPKD driver for paravirtualized NIC based on memory copy</td>
</tr>
<tr>
<td>virtio-net-pmd</td>
<td>DPKD driver for paravirtualized NIC based on Virtio</td>
</tr>
<tr>
<td>vmxnet3-usermap</td>
<td>DPKD driver for paravirtualized NIC in VMware ESXi</td>
</tr>
<tr>
<td>tools</td>
<td></td>
</tr>
<tr>
<td>dpdk-ci</td>
<td>Scripts for Continuous Integration</td>
</tr>
<tr>
<td>dpdk-web</td>
<td>Web pages</td>
</tr>
<tr>
<td>dts</td>
<td>DPKD Test Suite</td>
</tr>
<tr>
<td>stable-scripts</td>
<td>Scripts for stable branch maintenance</td>
</tr>
</tbody>
</table>
Agenda

A Quick Overview

Features Got Introduced in Recent Release, Nov. 2017

We Listened To You – Making it Easy to Use

Your Hands-On

Call For Action
Generic Flow API
The Generic Flow API (rte_flow) provides a common API for matching and filtering packets. Flow rules specific to the pattern to be matched, and the actions to be performed on packets when matches occur.
What about your investment on one NIC?
How much same look and feel?
How about reusability?
Generic Flow API

- Provides a generic way to configure hardware to match specific ingress or egress traffic, alter its fate and query related counters.
  - Don’t need to know the HW-specific filters

- Flow rules are composed of:
  - Attributes (properties of a flow rule such as its direction (ingress or egress) and priority)
  - Matching pattern (traffic properties to match)
  - Actions (operations to perform whenever a packet is matched by a pattern)

- Rule management:
  - rte_flow_validate()
  - rte_flow_create()
  - rte_flow_destroy()
  - rte_flow_flush()
Direct the VXLAN packet with specific inner MAC and VNI to queue #2.

### Legacy filter control API

```c
struct rte_eth_tunnel_filter_conf tunnel_filter_conf = {
    .outer_mac = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66};
    .inner_mac = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55};
    .inner_vlan = 0;
    .ip_type = RTE_TUNNEL_IPTYPE_IPV4;
    .ip_addr.ipv4_addr = 1;
    .filter_type = RTE_TUNNEL_FILTER_IMAC_TENID;
    .tunnel_type = RTE_TUNNEL_TYPE_VXLAN;
    .tenant_id = 1;
    .queue_id = 2;
};

int ret;

ret = rte_eth_dev_filter_ctrl(port_id, RTE_ETH_FILTER_TUNNEL,
    RTE_ETH_FILTER_ADD, &tunnel_filter_conf);
```

### Generic flow API

```c
const struct rte_flow_item pattern[] = {
    { RTE_FLOW_ITEM_TYPE_ETH, NULL, NULL, NULL},
    { RTE_FLOW_ITEM_TYPE_IPV4, NULL, NULL, NULL},
    { RTE_FLOW_ITEM_TYPE_UDP, NULL, NULL, NULL},
    { RTE_FLOW_ITEM_TYPE_VXLAN, { .vni = 1 }, NULL, { .vni = "\xff\xff\xff" }},
    { RTE_FLOW_ITEM_TYPE_ETH, 
        { .dst = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55 } }, NULL, 
        { .dst = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff } }},
    { RTE_FLOW_ITEM_TYPE_END, NULL, NULL, NULL},
};

const struct rte_flow_action actions[] = {
    { RTE_FLOW_ACTION_TYPE_PF, NULL},
    { RTE_FLOW_ACTION_TYPE_QUEUE, { .index = 2 }},
    { RTE_FLOW_ACTION_TYPE_END, NULL},
};

struct rte_flow_error flow_err;

flow_err = rte_flow_create(port_id, NULL, pattern, actions, &flow_err);
```
Browsing Driver .h files

**SoftNIC Driver header file**

http://dpdk.org/doc/api/rte__eth__softnic_8h_source.html

**SoftNIC Internals header file**

http://dpdk.org/doc/api/rte__eth__softnic__internals_8h_source.html
SoftNIC
Why SoftNIC? – Benefits

http://www.dpdk.org/ml/archives/dev/2017-October/078183.html

A software PMD that interfaces to a NIC PMD and provides software fallback for operations such as traffic management, metering, and policing, etc. that the underlying NIC does not support. This helps to make differences between NICs transparent to the application.
SoftNIC

- NIC operation executed by the CPU.
- Provides SW fallback for specific ethdev APIs:
  - Traffic Management (rte_tm)
  - Metering and Policing (rte_mtr) in release 18.02
  - Other capabilities to be added in future
- Generic: Works with any "hard" NIC.
- Configurable: User selects the features to be enabled.
- Helps to abstract the application from specific NIC capabilities by providing a software fallback if the NIC does not support the required function.

DPDK – Data Plane Development Kit

Copyright © 2018, Intel Corporation, All rights reserved.
*Other names and brands may be claimed as the property of others
Service Cores
Why Service Cores? – Benefits

* Service Cores provide a dynamic way of performing DPDK services on lcores.

* The mapping between service cores and services can be configured to abstract away the difference between platforms and environments, which simplifies application development.

* **Service Cores** – Browse rte_service.c

  http://dpdk.org/browse/dpdk/tree/lib/librte_eal/common/rte_service.c
Service Cores

Provides a dynamic way of performing DPDK services on lcores.

**Service**: A runnable work item, typically a DPDK function which requires an lcore (e.g., the SW eventdev PMD).

**Service Core**: A dedicated lcore for running services. Service cores are removed from the application’s coremask.

The mapping between service cores and services can be configured to abstract away the difference between platforms and environments.

For example, the Eventdev has hardware and software PMDs. Of these, the software PMD requires an lcore to perform the scheduling operations, while the hardware PMD does not. With service cores, the application would not directly notice that the scheduling is done in software.
* Service Cores – Sample Application

http://dpdk.org/browse/dpdk/tree/examples/service_cores/main.c

* Service Cores – API

http://dpdk.org/doc/api/rte__service_8h.html

* Service Cores – Source

http://dpdk.org/doc/api/rte__service_8h_source.html
Agenda

A Quick Overview

Features Got Introduced in Recent Release, Nov. 2017

We Listened To You – Making it Easy to Use

Your Hands-On

Call For Action
Hands-On
You Said – EASE of USE Is IMPORTANT TO YOU...
And ....
You Said – EASE of USE is IMPORTANT TO YOU...
And ....

We Listened!!
Quick start

The following script *(dpdk-quickstart.py)* provides a quick way to get DPDK up and running.

The script performs the following operations, on a guided and interactive graphical interface:
- Gets the latest DPDK release from the main git repository
- Checks and downloads the necessary dependencies
- Builds DPDK with the default target
- Sets up the minimum amount of hugepages to run TestPMD (a basic L2 forwarding application)
- Helps the user to select the ports to use on DPDK (including loopback detection between ports)
- Runs TestPMD, to show packet forwarding between two ports (virtual interfaces can also be used, if no ports connected with a loopback are available).

Note that this script is only supported on Fedora and Ubuntu.

Alternatively, DPDK can be installed and configured manually, with the following instructions:

"A simple forwarding test with pcap PMD which works with any NIC (with performance penalties)"

Extract sources.

```
tar xf dpdk.tar.gz
cd dpdk
```
## DPDK QUICK START

**Problem Statement:** DPDK lacks a user-friendly quick start guide

### Scope:

One script to:

- Check prerequisites
- Download and compile DPDK
- Configure the system
- Detect loopbacks
- Run TestPMD

### Progress to date:

- Script ready for upstreaming
- Patch set created
- Stretch goal of “loopback autodetect” implemented successfully

### Findings:

- Order of magnitude reduction in first run of a DPDK sample application

### Next Steps:

- Upstream patch to website to add script to DPDK quick start page

---

**Thanks to Team Members:** Pablo, Dave, Kirill, Herakliusz, Greg

---

Copyright © 2018, Intel Corporation, All rights reserved. *Other names and brands may be claimed as the property of others*
‘Dialog’-Based User Interface

• Same technology as ‘make menuconfig’ in kernel
• Easy-to-use dialog and menu system
• Guides the user through installation
• Gives sensible defaults along the way
Install PuTTY

- Click the URL
  
  http://the.earth.li/~sgtatham/putty/0.60/x86/

- Double-click putty-0.60-installer.exe option

- You have downloaded to your download folder

- Right-click and run as administrator

You have installed PuTTY

- Next step is to configure PuTTY
PuTTY Config: → Session Screen 1/2
1) Host name 207.108.8.161
2) Port 22
3) Connection Type SSH

1st select this
Keepalives – 30 sec.; Enable TCP Keepalives
If You Are Not on Intel Network
PuTTY Config: Connection -- > Proxy Screen 2/2

1) **Proxy Type**: None
2) **Proxy host name**: Leave it blank
3) **Do DNS name lookup at proxy end**: Auto

- Next step is to save the configuration
1) Name The Configuration. 2) Save It.

- Click “Session” on the top-left corner.
- It will take you to the 1st screen - shown here.
- In Saved Sessions Box, Enter a name, e.g., dpdk training cluster.
- Press Save button.
- From now on, you can load the saved session when starting PuTTY.
Getting to The HostVM-<m> Through Cluster Jump Server

1. Your Laptop

2. Cluster Jump Server

3. HostVM
Start PuTTY

Find PuTTY that you just installed.

Click on icon “putty.exe”. You will get the PuTTY Configuration screen.

Select the Saved Session, shown here e.g., dpdk training cluster.

Press Load button. Press Open button. You will get ssh session shown in 3rd screen asking username.

Username: student<m>  [<m> is given to you. For example, it may be student19 or student25 or any other student<m>]

Ask for your specific <m> and use only that. This will avoid overlapping with other teams' <m>.

Password is same as username. For example, if your username is student19, then password is student19.

Repeat the above steps so that you have many connections to the jump server.
How to Log in and Connect to <HostVM>

Log in as student<m>; password student<m>

The next step is to connect to ubuntu@<HostVM#> assigned to you.

In case it is not given, ask for your own <HostVM#>.

Note: We are using 1:1 – same student name and same HostVM#- name.

Type in the following command ssh to username ubuntu@<HostVM#>.

```
ssh ubuntu@<HostVM#>                <HostVM#>  dbdw01 to dbdw17
dskl01 to dslk10
```

Password: same as username above.

```
sudo su –
cd /home/ubuntu
```
Sudo python dpdk-quickstart.py

We are about to install following dependencies:
libnuma-devel
dialog
Do you want to continue? (Y/n)
Y

Reading package lists...
Press any key to continue...
Choose Virtual Ports
Side By Side Running 2 Servers
Loopback Test – output

Port statistics

RX-packets: 152333380  RX-missed: 85144  RX-bytes: 18023253628
RX-errors: 0
RX-nombuf: 0
TX-packets: 110972150  TX-errors: 0  TX-bytes: 20300138796

Throughput (since last show)
Rx-pps: 8074291
Tx-pps: 6617817

NIC statistics for port 1
RX-packets: 111018404  RX-missed: 12653  RX-bytes: 20307876440
RX-errors: 0
RX-nombuf: 0
TX-packets: 152336241  TX-errors: 0  TX-bytes: 18023613538

Throughput (since last show)
Rx-pps: 6620746
Tx-pps: 8074279
1) For Binding again to kernel driver

```
cd /home/ubuntu/dpdk_demo/usertools
./dpdk-devbind.py -status
./dpdk-devbind.py -bind=ixgbe 03:00.0
./dpdk-devbind.py -bind=ixgbe 03:00.1
ifconfig -a
ifconfig enp3s0f0 promisc up
ifconfig enp3s0f1 promisc up
```

2) To Start Again the script

```
cd /home/ubuntu
python dpdk-quickstart.py
```

Let Us Browse dpdk_demo, You Just Created!

```
root@dpdk:~$ ls -al
usr/local/bin
usr/local/man
usr/local/src
usr/local/include
usr/local/lib
usr/local/share
```

```
root@dpdk:~$ ls -al
usr/local/share/doc
dpdk-1.17.0/doc
```

```
root@dpdk:~$ ls -al
usr/local/share/doc
dpdk-1.17.0/doc
```

```
root@dpdk:~$ ls -al
usr/local/bin
usr/local/man
usr/local/src
usr/local/include
usr/local/lib
usr/local/share
```

```
root@dpdk:~$ ls -al
usr/local/share/doc
dpdk-1.17.0/doc
```

```
root@dpdk:~$ ls -al
usr/local/share/doc
dpdk-1.17.0/doc
```
How Many Microbenchmark Tests You Have?

/test/test$ ls -al
Quiz Time!! Does DPDK Have Scheduler?

What About Scalability?

Polling - No Scheduling!!
EventDev
Fig. 31.1: Sample eventdev usage, with RX, two atomic stages and a single-link to TX.

A high level overview of the setup steps are:

- `rte_event_dev_configure()`
- `rte_event_queue_setup()`
- `rte_event_port_setup()`
- `rte_event_port_link()`
- `rte_event_dev_start()`
EventDev

- Provides an event-driven programming model.
- Lcores call a scheduler that selects packets for them based on programmer-specified criteria.
- Benefits include efficient core utilization and dynamic load balancing.
- Supports the following scheduling types per queue:
  - Atomic (a flow can only be processed on one core at a time)
  - Ordered (a flow can be processed on two or more cores simultaneously, but packet order is maintained)
  - Parallel (all cores can process all flows, with no preservation of packet order)
Timer Event Adapter

- Can be used to connect timer events to an event device.
- Planned for release: 18.05.
Ethernet Event Adapter

- Service cores allow DPDK components to run on lcores with minimal application changes.

- Event adapters use service cores for SW-based transfers.

- The application configures the service core mapping.
31.1 Event struct

The eventdev API represents each event with a generic struct, which contains a payload and metadata required for scheduling by an eventdev. The \texttt{rte\_event} struct is a 16 byte C structure, defined in \texttt{libs/librte\_eventdev/rte\_eventdev.h}.
31.1.2 Event Payload

The rte_event struct contains a union for payload, allowing flexibility in what the actual event being scheduled is. The payload is a union of the following:

- `uint64_t u64`
- `void *event_ptr`
- `struct rte_mbuf *mbuf`

These three items in a union occupy the same 64 bits at the end of the rte_event structure. The application can utilize the 64 bits directly by accessing the u64 variable, while the event_ptr and mbuf are provided as convenience variables. For example the mbuf pointer in the union can used to schedule a DPDK packet.
31.1.1 Event Metadata

The rte_event structure contains the following metadata fields, which the application fills in to have the event scheduled as required:

- **flow_id** - The targeted flow identifier for the enq/deq operation.
- **event_type** - The source of this event, eg RTE_EVENT_TYPE_ETHDEV or CPU.
- **sub_event_type** - Distinguishes events inside the application, that have the same event_type (see above)
- **op** - This field takes one of the RTE_EVENT_OP_* values, and tells the eventdev about the status of the event - valid values are NEW, FORWARD or RELEASE.
- **sched_type** - Represents the type of scheduling that should be performed on this event, valid values are the RTE_SCHED_TYPE_ORDERED, ATOMIC and PARALLEL.
- **queue_id** - The identifier for the event queue that the event is sent to.
- **priority** - The priority of this event, see RTE_EVENT_DEV_PRIORITY.
31.2.1 Init and Config

The eventdev library uses vdev options to add devices to the DPDK application. The `--vdev` EAL option allows adding eventdev instances to your DPDK application, using the name of the eventdev PMD as an argument.

For example, to create an instance of the software eventdev scheduler, the following vdev arguments should be provided to the application EAL command line:

```
./dpdk_application --vdev="event_sw0"
```

In the following code, we configure eventdev instance with 3 queues and 6 ports as follows. The 3 queues consist of 2 Atomic and 1 Single-Link, while the 6 ports consist of 4 workers, 1 RX and 1 TX.

```c
const struct rte_event_dev_config config = {
    .nb_event_queues = 3,
    .nb_event_ports = 6,
    .nb_events_limit = 4096,
    .nb_event_queue_flows = 1024,
    .nb_event_port_dequeue_depth = 128,
    .nb_event_port_enqueue_depth = 128,
};
int err = rte_event_dev_configure(dev_id, &config);
```
Packet transfer between Ethernet devices and event devices can be supported in hardware or via a software thread (receiving packets from the ethdev port and enqueuing these as events to the event device).

The Event Ethernet Rx Adapter library allows application code to configure both transfer mechanisms using a common API.

Adapters will also be provided by crypto devices and for timers.
31.2.2 Setting up Queues

Once the eventdev itself is configured, the next step is to configure queues. This is done by setting the appropriate values in a queue_conf structure, and calling the setup function. Repeat this step for each queue, starting from 0 and ending at nb_event_queues - 1 from the event_dev config above.

```c
struct rte_event_queue_conf atomic_conf = {
    .schedule_type = RTE_SCHED_TYPE_ATOMIC,
    .priority = RTE_EVENT_DEV_PRIORITY_NORMAL,
    .nb_atomic_flows = 1024,
    .nb_atomic_order_sequences = 1024,
};

int dev_id = 0;
int queue_id = 0;
int err = rte_event_queue_setup(dev_id, queue_id, &atomic_conf);
```
31.2.3 Setting up Ports

Once queues are set up successfully, create the ports as required. Each port should be set up with its corresponding port_conf type, worker for worker cores, rx and tx for the RX and TX cores:

```c
struct rte_event_port_conf rx_conf = {
    .dequeue_depth = 128,
    .enqueue_depth = 128,
    .new_event_threshold = 1024,
};
struct rte_event_port_conf worker_conf = {
    .dequeue_depth = 16,
    .enqueue_depth = 64,
    .new_event_threshold = 4096,
};
struct rte_event_port_conf tx_conf = {
    .dequeue_depth = 128,
    .enqueue_depth = 128,
    .new_event_threshold = 4096,
};
int dev_id = 0;
int port_id = 0;
int err = rte_event_port_setup(dev_id, port_id, &CORE_FUNCTION_conf);
```

It is now assumed that:

- port 0: RX core
- ports 1,2,3,4: Workers
- port 5: TX core
31.2.4 Linking Queues and Ports

The final step is to “wire up” the ports to the queues. After this, the eventdev is capable of scheduling events, and when cores request work to do, the correct events are provided to that core. Note that the RX core takes input from eg: a NIC so it is not linked to any eventdev queues.

Linking all workers to atomic queues, and the TX core to the single-link queue can be achieved like this:

```c
uint8_t port_id = 0;
uint8_t atomic_qs[] = {0, 1};
uint8_t single_link_q = 2;
uint8_t tx_port_id = 5;
uint8_t priority = RTE_EVENT_DEV_PRIORITY_NORMAL;

for(int i = 0; i < 4; i++) {
    int worker_port = i + 1;
    int links_made = rte_event_port_link(dev_id, worker_port, atomic_qs, NULL, 2);
}
int links_made = rte_event_port_link(dev_id, tx_port_id, &single_link_q, &priority, 1);
```
31.2.5 Starting the EventDev

A single function call tells the eventdev instance to start processing events. Note that all queues must be linked to for the instance to start, as if any queue is not linked to, enqueuing to that queue will cause the application to backpressure and eventually stall due to no space in the eventdev.

```c
int err = rte_event_dev_start(dev_id);
```
31.2.6 Ingress of New Events

Now that the eventdev is set up, and ready to receive events, the RX core must enqueue some events into the system for it to schedule. The events to be scheduled are ordinary DPDK packets, received from an `eth_rx_burst()` as normal. The following code shows how those packets can be enqueued into the eventdev:

```c
const uint16_t nb_rx = rte_eth_rx_burst(eth_port, 0, mbufs, BATCH_SIZE);

for (i = 0; i < nb_rx; i++) {
    ev[i].flow_id = mbufs[i]->hash.rss;
    ev[i].op = RTE_EVENT_OP_NEW;
    ev[i].sched_type = RTE_SCHED_TYPE_ATOMIC;
    ev[i].queue_id = 0;
    ev[i].event_type = RTE_EVENT_TYPE_ETHDEV;
    ev[i].sub_event_type = 0;
    ev[i].priority = RTE_EVENT_DEV_PRIORITY_NORMAL;
    ev[i].mbuf = mbufs[i];
}

const int nb_tx = rte_event_enqueue_burst(dev_id, port_id, ev, nb_rx);
if (nb_tx != nb_rx) {
    for (i = nb_tx; i < nb_rx; i++)
        rte_pktmbuf_free(mbufs[i]);
}
```
31.2.7 Forwarding of Events

Now that the RX core has injected events, there is work to be done by the workers. Note that each worker will dequeue as many events as it can in a burst, process each one individually, and then burst the packets back into the eventdev.

The worker can lookup the events source from `event.queue_id`, which should indicate to the worker what workload needs to be performed on the event. Once done, the worker can update the `event.queue_id` to a new value, to send the event to the next stage in the pipeline.

```c
int timeout = 0;
struct rte_event events[BATCH_SIZE];
uint16_t nb_rx = rte_event_dequeue_burst(dev_id, worker_port_id, events, BATCH_SIZE, timeout);

for (i = 0; i < nb_rx; i++) {
    /* process mbuf using events[i].queue_id as pipeline stage */
    struct rte_mbuf *mbuf = events[i].mbuf;
    /* Send event to next stage in pipeline */
    events[i].queue_id++;
}

uint16_t nb_tx = rte_event_enqueue_burst(dev_id, port_id, events, nb_rx);
```
31.2.8 Egress of Events

Finally, when the packet is ready for egress or needs to be dropped, we need to inform the eventdev that the packet is no longer being handled by the application. This can be done by calling `dequeue()` or `dequeue_burst()`, which indicates that the previous burst of packets is no longer in use by the application.

An event driven worker thread has following typical workflow on fastpath:

```c
while (1) {
    rte_event_dequeue_burst(...);
    (event processing)
    rte_event_enqueue_burst(...);
}
```
The cryptodev API supports hardware and software acceleration of symmetric crypto algorithms. Hardware acceleration is supported by Intel® QuickAssist Technology. Software acceleration is provided by the Intel® Multi-Buffer Crypto for IPsec Library and other optimized software libraries.
CryptoDev Overview

- Framework for processing symmetric crypto workloads. Consists of:
  - Crypto Poll Mode Drivers for hardware-accelerated (Intel® QuickAssist Technology) and software-based crypto primitives.
  - A standard API for all crypto PMDs.
- Effortless migration between hardware and software crypto, and between physical and virtual environments.
- Will be extended to support asymmetric crypto algorithms in future.

QAT = Intel® QuickAssist Technology

user application

DPDK CRYPTODEV API COMPONENTS

Symmetric Session Management
Operation Processing
Enqueue/Dequeue
Device Capabilities
Operation Provisioning
Device Statistics
Symmetric Algorithms Definitions
Symmetric Session Management
Queue Pair Management
Device Management
Operation Processing Enqueue/Dequeue

Crypto PMDs

QAT*
AESNI-MB
AESNI-GCM
KASUMI
SNOW3G
ZUC
OPENSSL
SCHEDULER

Intel® QAT Hardware
Intel® performance libraries
libssso.a

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

Copyright © 2018, Intel Corporation, All rights reserved.
CryptoDev History

- CryptoDev API introduced.
- PMD for Intel® QuickAssist Technology.
- PMD for Intel® AES-NI Multi-Buffer software library.

2.2

- Added Intel® AES-NI GCM PMD
- Support for SNOW 3G algorithm.
- Discovery mechanism.
- NULL PMD.

16.04

- HW support for 3DES, KASUMI, and NULL.
- SW support for ZUC (EEA3 and EIA3).
- OpenSSL PMD supporting multiple crypto algorithms.
- Supports all “MUST” and “SHOULD+” algorithms for IPsec (ESP).

16.07

- Optimized AES-GCM SW PMD.
- Support for chained mbufs.
- Round Robin Scheduler.

16.11

- Packet-based scheduler.
- Support for cable (DOCSIS) algorithms.
- CryptoDev and VPP used to demonstrate 100Gbps IPsec performance on Purley platform.

17.02

17.05

Copyright © 2018, Intel Corporation, All rights reserved.
*Other names and brands may be claimed as the property of others
CryptoDev Packet Processing Flow

DPDK Application

Application Code

ETHDEV API

I40E PMD

CRYPTODEV API

SW Crypto
PMD

QAT PMD

CRYPTODEV API

ETHDEV API

I40E PMD

Application Code

HW/SW Boundary

DPDK API

I40E PMD

Encrypted packet flow (encryption)

Plaintext packet flow (encryption)

DPDK – Data Plane Development Kit

Copyright © 2018, Intel Corporation, All rights reserved.
*Other names and brands may be claimed as the property of others
## Supported Algorithms

<table>
<thead>
<tr>
<th>Cipher Algorithms</th>
<th>Hash Algorithms</th>
<th>AEAD Algorithms</th>
</tr>
</thead>
</table>
| • AES CBC/CTR 128/192/256 bit  
• Snow3G (UEA2)  
• KASUMI F8  
• ZUC EEA3  
• AES_CFB | • MD5_HMAC*  
• SHA1/224*/256/384*/512  
• AES XCBC  
• Snow3G UIA2  
• KASUMI F9  
• ZUC EIA3  
• NULL | • AES GCM 128/192**/256 bit |

*Other names and brands may be claimed as the property of others
DPDK CryptoDev and Kernel Drivers

DPDK Application

Ethdev API
- I40E PMD
- VFIO / IGB_UIO
- PF
- FVL
- Intel® VT-d

CryptoDev API
- SW PMD
- QAT PMD
- VFIO / IGB_UIO
- PF
- VF
- Intel® QuickAssist Technology Accelerator

Application Code
- DPDK API
- DPDK Userspace Poll Mode Driver
- Kernel Device Driver
- Physical Device

Intel® QuickAssist Technology Accelerator

Copyright © 2018, Intel Corporation, All rights reserved.
*Other names and brands may be claimed as the property of others
Crypto Scheduler PMD

- Distributes crypto ops to multiple crypto PMDs (slaves).
- Supports multiple distribution modes:
  - Round-robin mode to balance workload across multiple slaves (DPDK 17.02)
  - Packet size mode (DPDK 17.05)
  - More modes are planned for future releases
- Provides API to manage slaves, set modes, and enable/disable ordering.
- Provides API for user to use their own custom mode.
IPsec

* IPsec acceleration uses the cryptodev API to take advantage of hardware and software crypto acceleration.
* A security gateway sample application exists in DPDK to demonstrate usage of the API for IPsec.
* A complete implementation exists in the Vector Packet Processing (VPP) project in FD.io.
* A new security API (rte_security) has also been recently introduced to manage hardware offloads for security protocols like IPsec.
IPsec Overview

- IPsec is a layer 3 IP security service.
- Security services offered by IPsec include:
  - Connectionless integrity
  - Data confidentiality (encryption)
  - Sequence integrity (partial, anti-replay windowing)
  - Limited traffic flow confidentiality (tunnel mode)
- These security services are provided by the use of two traffic security protocols:
  - Authentication Header (AH)
  - Encapsulating Security Payload (ESP)
- IPsec is designed to be crypto algorithm-independent.
DPDK IPSEC SECURITY GATEWAY SAMPLE APP
Provides an example of an IPsec security gateway.

Security Policies and Security Associations are manually configured.

The Security Policies (SP) are implemented as ACL rules. The Security Associations (SA) are stored in a table. Routing is implemented using LPM.

Hardware and software crypto acceleration are both supported.

Further info is available in the Sample Applications User Guide.
VECTOR PACKET PROCESSING (VPP) IPSEC
Cryptodev Integration with VPP

- The Vector Packet Processing (VPP) project in FD.io supports IPv4/IPv6 IPsec ESP, tunnel and transport mode, and SA management.
- The DPDK Cryptodev framework has been integrated into VPP to accelerate IPsec performance using Intel® Performance Libraries and/or Intel® QuickAssist Technology.
- This provides a complete IPsec solution, which demonstrates the performance that can be achieved on Intel® platforms.
VPP Overview

Fast, Scalable and Deterministic:
- 14+ Mpps per core
- Tested to 1 Tbps
- Scalable routing/forwarding tables, supporting millions of concurrent entries
- 0 packet drops, ~15μs latency

Optimized:
- DPDK for fast I/O
- ISA: SSE, AVX, AVX2
- IPC: Batching, no mode switching, no context switches, non-blocking
- Multi-core: Cache and memory efficient

Fully featured:
- L2: VLAN, Q-in-Q, Bridge Domains, LLDP ...
- L3: IPv4, GRE, VXLAN, DHCP, IPsec ...
- L3: IPv6, Discovery, Segment Routing ...
- L4: TCP, UDP ...
- CP: API, CLI, IKEv2 ...

Usability:
- Modular, graph-based architecture
- Language bindings
- Open Stack/ODL (Netconf/Yang)
- Kubernetes/Flanel (Python API)
- OSV packaging
Accelerated VPP IPsec

- Replaced VPP esp-encrypt and esp-decrypt nodes with dpdk-esp-encrypt and dpdk-esp-decrypt, which interface to Cryptodev.

- Also added the following nodes:
  - dpdk-crypto-input: polling input node, dequeuing from crypto PMDs
  - dpdk-esp-encrypt-post and dpdk-esp-decrypt-post: encapsulate to valid packet vectors from dequeued packets
SECURITY API AND IPSEC HARDWARE ACCELERATION
Security Acceleration API (rte_security)

- Framework for management and provisioning of hardware acceleration of security protocols.
- Generic APIs to manage security sessions.
- Security acceleration functions are accessed through security instances which can be instantiated on any device type. Currently supports security instances on Crypto and Ethernet devices.
- Rich capabilities discovery APIs.
- Currently only targets the support of the IP Security (IPsec) protocol, but is designed to be able to support other protocols in future (e.g., PDCP, SSL).
Security Acceleration Types

Supports three security acceleration types:

**Lookaside:** Packet is given to an accelerator for processing and then returned to the host after processing is complete.

**Inline Crypto:** Acceleration performed on the physical interface as packet ingresses/egresses the system.

  - No packet header modifications performed by the hardware, only encryption/decryption and authentication operations.

**Inline Protocol:** Acceleration performed on the physical interface as the packet ingresses/egresses the system.

  - Packet header modification is performed on hardware including all state management and encryption/decryption and authentication operations.
Lookaside IPsec Acceleration (Ingress)
Lookaside IPsec Acceleration (Egress)
Inline Crypto Acceleration (Ingress)
Inline Crypto Acceleration (Egress)
Homework

Choose 82599 Port 0 and Port 1
IP Link – Find the NICs You Have

1) ip link - to find the NIC devices you have
2) cat /sys/class/net/*/address – the list of addresses or
3) ethtool – P <eth>
4) Put the ports to be tested in promiscuous mode

```bash
ubuntu@dsk102:~$ sudo ifconfig eth0 promisc [sudo] password for ubuntu:
```

```bash
ubuntu@dsk102:~$ sudo ifconfig eth1 promisc
```

```bash
ubuntu@dsk102:~$ sudo ifconfig eth4 promisc
```

```bash
ubuntu@dsk102:~$ sudo ifconfig eth5 promisc
```
Loopback Test – output

1) For Binding again to kernel driver

```
cd /home/ubuntu/dpdk_demo/usertools
./dpdk-devbind.py -status
```

```
./dpdk-devbind.py -bind=ixgbe 03:00.0
```

```
./dpdk-devbind.py -bind=ixgbe 03:00.1
```

```
ifconfig -a
```

```
ifconfig enp3s0f0 promisc up
ifconfig enp3s0f1 promisc up
```


2) To Start Again the script

```
cd /home/ubuntu
python dpdk-quickstart.py
```
export TERM=putty

Before

And after...
Backup
ETSI NFVI Architecture – Intel® Select Fast Track Kit Mapping
ETSI NFVI Architecture – Intel® Select Fast Track Kit Mapping
Building Blocks – Intel® Select Fast Track Kit

High-Performing NFVI

Hardware Configuration
- Intel® Xeon® Gold processors
- Intel® 628 chipset with Intel® QAT
- Intel® Ethernet Converged Network Adapters

Drivers
- NIC, DPDK, Intel® QAT, etc.

Combination of OS Host/Guest
- Red Hat Enterprise Linux (RHEL)
- Ubuntu
- CentOS

Combinations of Hypervisor
- KVM/QEMU

Combinations of Virtual Networking I/O
- Combinations eliminated through use of “platform I/O services” concept
Two Targets For Conformity – 1) Base, 2) Plus
References

https://github.com/cisco-system-traffic-generator/trex-stateless-gui/releases/tag/v4.0

RFC2544
https://gerrit.fd.io/r/gitweb?p=csit.git;a=tree;f=resources;hb=HEAD

How to Install Grafana on Ubuntu 16.04 and Pull In home Assistant Data from InfluxDB

https://www.youtube.com/watch?v=oexrAKLQ_LI

InfluxDB – Ubuntu wget

wget https://dl.influxdata.com/influxdb/releases/influxdb_1.3.7_amd64.deb

sudo dpkg -i influxdb_1.3.7_amd64.deb

InfluxDB – Windows wget

unzip influxdb-1.3.7_windows_amd64.zip
Thanks!