SFO15-102: ODP Project Update
Topics for Today

- OpenDataPlane overview and project status
- Key developments since HKG15
- Plans for rest of 2015 and into 2016
- ODP and OPNFV
- ODP Application Design
- LNG Sessions and Demos at SFO15
ODP Overview
What is OpenDataPlane?

- **ODP API Specification**: An Abstract API Specification
- **ODP Implementations**: Multiple independently maintained implementations of the ODP API

Validation Test Suite
The ODP API Specification

- Open Source, open contribution, BSD-3 licensed
- Vendor and platform neutral
- Application-centric—covers functional needs of data plane applications
- Ensures portability by specifying functional behavior of ODP
- Defined jointly and openly by application writers and platform implementers
- Architected to be implementable on a wide range of platforms efficiently
- Sponsored, Governed, and Maintained by Linaro Networking Group (LNG)
ODP Implementations

Multiple independently maintained implementations of the ODP API

- One size does not fit all—widely differing internals among platforms
- Anyone can create an ODP implementation tailored to their platform
- Distribution and maintenance of each implementation as owner wishes
  - Open source or closed source as business needs determine
  - Have independent release cycles and service streams
- Allows HW and SW innovation in how ODP APIs are implemented on each platform
LNG distributes and maintains a number of Reference Implementations of ODP

- Open source, open contribution, BSD-3 licensed
- Provide easy bootstrapping of ODP onto new platforms
- Implementers free to borrow or tailor code as needed for their platform
- Implementers retain full control over their own implementations whether or not they are derived from a reference implementation
ODP Validation Test Suite

- Synchronized with ODP API Specification level
- Maintained and distributed by LNG
- Open source, open contribution, BSD-3 licensed
- Key to ensuring application portability across all ODP implementations
- Tests that implementations of ODP conform to the specified functional behavior of ODP APIs
- Can be run at any time by both users and vendors to validate implementations of ODP

Validation Test Suite
ODP Project Status
ODP Project History

- October, 2013: Announced at LCU ‘13
- 2014: Preview releases (v0.1 - v0.11)
- **2015: Pre-Production Evaluation releases**
  - February: ODP v1.0
  - May: ODP v1.1
  - July: ODP v1.2
  - **September: ODP v1.3**
  - ODP v1.4 and v1.5 planned for later this year
  - Monarch Release (EOY 2015)
- **2016: Production releases**
  - Focus on apps and OPNFV
Key API Changes Since HKG15

ODP v1.1:
- User metadata support

ODP v1.2:
- Streamlined pool creation
- Added default cpumasks and introduced thread masks (thrmask) and thread types (worker, control)
Key API Changes Since HKG15 (Cont’d)

ODP v1.3:

- PktIO improvements: Start/Stop, parameterization, input/output modes
- Parameter structure init routines
- Scheduler improvements: Groups, prefetching
- Ordered queues
- Ordered locks
Other ODP Changes since HKG15

Platforms

- odp-dpdk launched as performance platform for x86
  - uses DPDK as underlying SDK
  - enables ODP to support various NICs that have DPDK drivers
  - doing proof of concept work with applications like Open vSwitch
- odp-mppa launched as open source platform supporting Kalray MPPA SoCs
Other ODP Changes since HKG15 (Cont’d)

Validation Suite

- Restructured to be platform-independent
- Supports platform-specific extension tests that can be added by vendors for additional internal validation
- Goal is to support validation repository separation as described earlier
What’s ahead for ODP in 2015?

ODP v1.4:

- Egress Traffic Manager
- Interprocess Communication (IPC)
- Port counters and statistics
- Improved test capabilities (PCAP replay)
- Configuration APIs
What’s ahead for ODP in 2015? (Cont’d)

ODP v1.5

- Generic NIC driver interface
- Integration of RSS with Classifier and Scheduler

Monarch Release

- Transition to production release cycles and packaging (RPM, Debian), formal use of Release Candidates (RCs)
ODP and OPNFV
What is OPNFV?

From OPNFV.org:

OPNFV is a carrier-grade, integrated, open source platform to accelerate the introduction of new NFV products and services.

In Practice

OPNFV aims to realize the ETSI NFV vision.
DPACC Project and Terminology

**g-API**
- General API used by Virtual Network Functions (VNFs)

**s-API**
- System API representing a platform’s native capabilities
ODP and DPACC

Both ODP and DPDK proposing themselves as DPACC’s g-API layer

Discussions ongoing as to how the two projects can cooperate on future architectural direction

Issues revolve around application design philosophy
ODP Application Design
ODP Application Design Focus Areas

- Portability
- Transparent access to platform acceleration and offload
- Scalability

Also need to consider migration for existing applications
ODP Approach to Portability

Abstract API Design

- Use of abstract types (e.g., `odp_packet_t`) rather than platform-specific structs
- API selections reflect application use cases but are balanced against ability to be mapped efficiently to widely varying platform capabilities
  - Not interested in “least common denominator” APIs
  - Not interested in overly high-level APIs that few if any platforms can support efficiently
  - Expect APIs to evolve over time
ODP Approach to Acceleration

Be able to exploit platform-specific acceleration and offload capabilities (HW and SW) without application effort

- Abstract APIs are efficiently mappable directly to platform HW capabilities, e.g.:
  - HW buffer/packet mgmt
  - Integrated I/O adapters
  - HW parsing and classification
  - HW scheduling and flow ordering
  - HW egress traffic shaping and QoS, etc.
ODP Approach to Scalability

Support scalability to many-core architectures without application redesign.

- Classifier
- Events
- Queues
- Scheduler
- Traffic Manager

No changes to application design when running on 4, 40, or 400 cores
ODP Concepts: Packet Receive

Ethernet → PktIO → Classifier → Class of Service (CoS) → PMRs → Queue

Packet storage → Input to Scheduler
ODP Concepts: Event Scheduling

Queues store events

Threads call odp_schedule() to get next event

Threads invoke engines via ODP APIs

Implementations “wrapper” engines to interact with rest of ODP infrastructure

Threads call odp_queue_enq() to add events to queues to be scheduled for further processing
ODP Queue Scheduling Attributes

- None (Parallel)
  - Events processed independently by multiple threads

- Atomic
  - Events serialized by scheduler, so no locks needed

- Ordered
  - Events scheduled in parallel, with order preservation
  - Threads can use ordered locks for ordered critical sections within parallel flow processing
Parallel Processing of Single Flows

Scheduler dispatches events from ordered queues to eligible worker threads concurrently.

Worker threads process events in parallel, use ordered critical sections as needed.

Processed events appear on output queue in same order as the originating ordered queue.
Simplified worker thread structure

```c
void worker_thread(...) {
    odp_init_local(ODP_THREAD_WORKER) /* And other init processing */
    while (1) {
        ev = odp_schedule() /* Get next event to be processed */
        ...process work in parallel with other threads
        odp_schedule_order_lock() /* Enter ordered critical section */
        ...critical section processed in order
        odp_schedule_order_unlock() /* Exit ordered critical section */
        ...additional work processed in parallel with other threads
        odp_queue_enq(queue, ev) /* Send event to next processing stage */
    }
}
```
ODP Concepts: Traffic Manager

Thread

... Thread

Thread

ODP Traffic Manager (coming in ODP v1.4)

TM Input Queues

Arbiters, Shapers, etc.

TM Output Queues

TM Configuration APIs

PktIO

Ethernet

Ethernet

Loopback

Back to Ingress
Simplified worker thread structure (with TM)

void worker_thread(...) {
    odp_init_local(OPD_THREAD_WORKER) /* And other init processing */
    while (1) {
        ev = odp_schedule() /* Get next event to be processed */
        ...process work in parallel with other threads
        odp_schedule_order_lock() /* Enter ordered critical section */
        ...critical section processed in order
        odp_schedule_order_unlock() /* Exit ordered critical section */
        ...additional work processed in parallel with other threads
        odp_tm_enq(tm_queue, pkt) /* Output Packet via Traffic Manager */
    }
}
Application Staged Migration Path to ODP

- **Legacy App**
  - Uses SDK or RYO APIs
  - Directly tied to specific HW platform
  - I/O via PMDs

- **Initial port**
  - Uses mix of ODP and legacy APIs
  - Semi-portable to other platforms, limited acceleration
  - I/O via ODP packet I/O (poll mode) APIs

- **ODP App**
  - Uses ODP APIs
  - Fully portable with transparent platform-specific acceleration
  - Restructured to use ODP event scheduler for processing and I/O scalability

- **Native port**
  - Uses ODP APIs
  - Fully portable with transparent platform-specific acceleration
  - Restructured to use ODP event scheduler for processing and I/O scalability
LNG Sessions and Demos at SFO15

OpenDataPlane.org
LNG Sessions and Demos at SFO15

Thursday

● SFO15-400: Core Isolation
● SFO15-404: ODP User Experience Roundtable
● SFO15-408: ODP-DPDK Platform Performance

Demo Thursday

● Demos by Broadcom, Cavium, EZchip, Freescale, HiSilicon, and LNG
Thank you