BKK16-106:
ODP Project Update
Topics for Today

- OpenDataPlane overview and project status
- Key developments since SFO15
- ODP Directions in 2016
- ODP Application Design
- LNG Sessions and Demos at BKK16
What is OpenDataPlane?

- ODP API Specification
- 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 (depends only on C99)
- 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
ODP Implementations (Cont’d)

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
## LNG Reference ODP Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Owner/Maintainer</th>
<th>Target Platform</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux-generic</td>
<td>Open contribution, maintained by LNG</td>
<td>Pure SW, runs on any Linux kernel. Functional implementation, not a performance target, but has been a testbed for performance enhancements.</td>
<td>Any</td>
</tr>
<tr>
<td>odp-dpdk</td>
<td>Open contribution, developed by LNG</td>
<td>Intel x86 using DPDK as SW acceleration layer</td>
<td>Intel x86</td>
</tr>
</tbody>
</table>
# Commercial ODP Implementations

<table>
<thead>
<tr>
<th>Owner/Maintainer</th>
<th>Target Platform(s)</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcom</td>
<td>BMC57812</td>
<td>ARMv8</td>
</tr>
<tr>
<td>Cavium Networks</td>
<td>Octeon, ThunderX</td>
<td>MIPS64, ARMv8</td>
</tr>
<tr>
<td>EZchip (now Mellanox)</td>
<td>TileGx SoC</td>
<td>TileGx</td>
</tr>
<tr>
<td>Freescale (now NXP)</td>
<td>QorIQ SoCs</td>
<td>Power, ARMv8</td>
</tr>
<tr>
<td>Huawei / HiSilicon</td>
<td>D02 Board</td>
<td>ARMv8</td>
</tr>
<tr>
<td>Kalray</td>
<td>KONIC-80 SoC</td>
<td>MPPA (Proprietary)</td>
</tr>
<tr>
<td>Texas Instruments</td>
<td>Keystone II</td>
<td>ARM Cortex A15</td>
</tr>
</tbody>
</table>
ODP Sponsorship and Governance

- The Linaro Networking Group and its 13 member companies are sponsors and upstream maintainers of ODP.
- LNG membership is open to all.
- ODP is fully open source and open contribution, uses BSD 3-clause licensing.
- All ODP design work is carried out in public with both open face-to-face meetings and weekly public architecture calls, and on the ODP mailing list.
ODP Project History

- October, 2013: Announced at Linaro Connect USA ‘13
- 2014: Preview releases (v0.1 - v0.11)
- 2015: Pre-Production Evaluation releases (v1.0 - v1.6)

- **2016: Production releases**
  - February: ODP v1.7
  - Production Release RC1 (Monarch) Today
  - Production Release RC2 (Monarch) April
  - Production Release (Monarch) 2Q16 - Long Term Support (LTS)
  - Focus on apps and OPNFV
Key ODP Developments Since SFO15

Functionality

- Improved time, crypto, classification, synchronization/atomics, threading, initialization/termination, configuration, queues, pktio, API standardization

Performance

- Multi-buffer alloc-free
- Multi-queue I/O

Documentation

- Improved User Guide and Implementation Guide
ODP Path to Monarch

Monarch RC1 (Today)

- Existing APIs frozen. A few additional APIs still under final negotiation in areas of Traffic Manager, Packet Composites, Queue Groups, NUMA.

Monarch RC2 (April 2016)

- All APIs frozen. Implementations for linux-generic and odp-dpdk available. Validation suite and Documentation updated.

Monarch (2Q16)

- At least two SoC implementations available
- Implementations will be supported for at least 1 year (LTS)
ODP Application Design Focus Areas

- Portability
- Transparent access to platform acceleration and offload
- Scalability (event-driven application model)
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.

- Threads
- Events
- Queues
- Scheduler
- PktIO
- Pktin/Pktout Queues
- Classifier
- Traffic Manager

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

Control Thread
- Housekeeping and general tasks
- Share resources with Linux

Worker Thread
- Packet processing
- Use core isolation for best performance

Threads process EVENTS
ODP Concepts: Event

Data Events
- Buffer
- Packet

System Events
- Completion notification
- Timer expiration

Events stored in QUEUEs
ODP Concepts: Queue

Plain Queue
- Independent Enq/Deq operations (single and multi)
- MT Safe or MT Unsafe
- Application private use
- Used by PktIO objects in QUEUE mode

Scheduled Queue
- Apps Enq, ODP Scheduler can Deq
- Always MT Safe
- Automatic load balancing and scaling
- Automatic queue context synchronization services
- Used by PktIO objects in SCHED mode
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

- **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 Queues and Flow Processing

Scheduler dispatches events from parallel queues to threads individually.

Worker threads process events in parallel, any synchronization needed among events is application responsibility.

Processed events appear on output queue in unpredictable order.
Atomic Queues and Flow Processing

Scheduler dispatches events from atomic queues to threads individually.

Worker threads process events in parallel, scheduler ensures no two threads can process events from the same atomic queue at the same time.

Processed events appear on output queue in same order as the originating atomic queues because scheduler has serialized them.
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.
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: PktIO

Input (in_mode)

- Direct: Application reads packets directly from device input queue
- Queue: Application reads packets from device event queues
- Schedule: Application receives packets via scheduler
- Disabled: Input disabled

Output (out_mode)

- Direct: Application writes packets directly to device output queue
- Queue: Application writes packets to device event queues
- TM: Application writes packets to Interface via Traffic Manager
- Disabled: Output disabled
ODP Concepts: Direct Packet Receive

PktIO opened with in_mode = ODP_PKTIN_MODE_DIRECT

Optional Hashing on IPv4/v6 and TCP/UDP fields

Associate receive pktin queue with each receiver thread
ODP Concepts: Direct Packet Send

odp_pktout_send()

Queue

PktIO

Ethernet

PktIO opened with out_mode = ODP_PKTOUT_MODE_DIRECT

Queue

Queue

Queue

Pktout queues created and configured with odp_pktout_queue_config()
ODP Concepts: Packet Receive via Queues

- Ethernet
- PktIO
- Hash

PktIO opened with in_mode = ODP_PKTIN_MODE_QUEUE
Optional Hashing on IPv4/v6 and TCP/UDP fields

Associate receive queue with each receiver thread

- `odp_pktin_queue_config()`
- `odp_queue_deq()`
- `odp_queue_enq()`
ODP Concepts: Transmit via Queues

Thread

Thread

Thread

Thread

odp_queue_enq()

Queue

Queue

Queue

Queue

PktIO

Ethernet

PktIO opened with out_mode = ODP_PKTOUT_MODE_QUEUE

Output queues created and configured with
odp_pktout_queue_config()
PktIO opened with `in_mode = ODP_PKTIN_MODE_SCHED`

Optional Hashing on IPv4/v6 and TCP/UDP fields

Distribute packets to multiple queues for scheduling

```c
odp_pktin_queue_config()
```
ODP Concepts: Packet RX via Classifier

- Ethernet
- ... Ethernet
- Ethernet

Packet storage

Scheduler

Input to Scheduler

PktIO opened with in_mode = ODP_PKTIO_MODE_SCHED

Associate RX queues with classifier input

Use RX queues or create additional queues for CoS use

- odp_pktin_queue_config()
- odp_pktin_event_queue(), odp_queue_create()
- odp_cls_cos_create()
ODP Concepts: Traffic Manager

Thread

Thread

Thread

Ethernet

Ethernet

Loopback

Back to Ingress

ODP Traffic Manager (coming in ODP Monarch)

TM Input Queues

Arbiters, Shapers, etc.

TM Output Queues

TM Configuration APIs

PktIO

PktIO opened with out_mode = ODP_PKTIO_MODE_TM
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_tm_enq(tm_queue, pkt) /* Output Packet via Traffic Manager */
    }
}
ODP Application Design Summary

Poll Mode Design
- Compatibility with existing application design pattern
- Facilitates porting to ODP
- Queued PktIO model provides enhanced performance
- Queued PktIO model allows use of classifier

Event Mode Design
- Automatic scalability to many core systems
- Integrated with Classifier for additional flexibility and offload
- Simplified synchronization via automatic queue context management
- Preferred model for new applications
LNG Sessions and Demos at BKK16

OpenDataPlane.org
Thursday LNG Sessions at BKK16

BKK16-401: Enhancing Application Performance with ODP

BKK16-405: LNG Future Directions

BKK16-409: VOSYSwitch port to ARMv8 Platforms and ODP Integration
OpenFastPath

- Open source, optimized IP stack built on ODP
- Hosted on GitHub
- Supports IPv4, IPv6, TCP, and UDP
- Supports GRE and VXLAN tunnelling
- More information at OpenFastPath.org
● Open source high performance HTTP server
● Also can serve as a reverse proxy server, mail proxy server, and TCP proxy server
● Ported to run with OFP on ODP
● More information at nginx.org
• Open source stateful traffic generator
• Developed by Cisco, Hosted on GitHub
• Supports up to 40Gb Interfaces
• DPDK application ported to ODP
• More information at trex-tgn.cisco.com
VOSYSwitch is a commercial switch product based on Snabb Switch
Being ported to ODP
More information at virtualopensystems.com
Thank you

OpenDataPlane.org