DPDK

DATA PLANE DEVELOPMENT KIT

Userspace 2015 | Dublin

DPDK Integration: A Product's Journey

Agenda

- Introduction & Background
- Implementation:
  - Experimenting with DPDK
  - Observations and Lessons Learned
  - Moving Forward
- Q&A

DPDK

Introduction & Background

Cisco IOS-XE Data Plane

- Large shared code base with a common architecture
- Runs on a wide variety of environments:
  - Cisco custom ASICS
  - Commercial multicore processors
  - Commercial multicore processors within a hypervisor
- Performance spectrum ranges from 50 Mbits/sec to 200 Gbits/sec
- Legacy and modern tool chains: Open source GCC, Custom GCC, Intel ICC, CLANG
- Legacy kernel 2.6.3x and forward
- 32bit and 64bit applications
- Scales from 1 to 1000s of cores
- Resides deep in a layered system

Experimenting with DPDK

Our Interest in DPDK

We're intrigued by DPDKs potential benefits:

- Performance in general
- Reduced system calls
- Simplified I/O batching
- Zero ISR core utilization
- Driver debugging
- Memory isolation
- Less risk of kernel crash

DPDK Integration

DPDK
DATA PLANE DEVELOPMENT KIT
Userspace 2015

Phase 1: Data Plane only Integration

- Integrate DPDK into our DP:
  - I/O
  - Memory and buffer management
  - Thread management where required by DPDK
- Fedora build and runtime environment:
  - We built DPDK libs the normal way
  - We built our DP using host tools and libraries
  - We linked our DP to DPDK dynamic libs

Phase 2: Full Integration

- Integrate DPDK into our build environment
- Integrate DPDK into our runtime environment
- Enable management of DPDK interfaces into our control plane

DPDK

Implementation Observations

Implementation Observations

Observation	Description	Solution/Workaround
Adapting multi-threaded applications with more threads than physical cores can be difficult	Our application is multi-threaded and must scale down to a single core, but early versions of DPDK enforced a 1:1 mapping between lcores and physical cores.	From DPDK 2.0 forward, multi-lcore/ physical core is supported.
Multi-socket VM initialization failures on legacy kernels	Configuring multiple virtual sockets in a VM, particularly with legacy kernels, could result in NUMA panic.	Issue addressed in DPDK 2.0 and forward by removing the fall back to the physical_device_id.

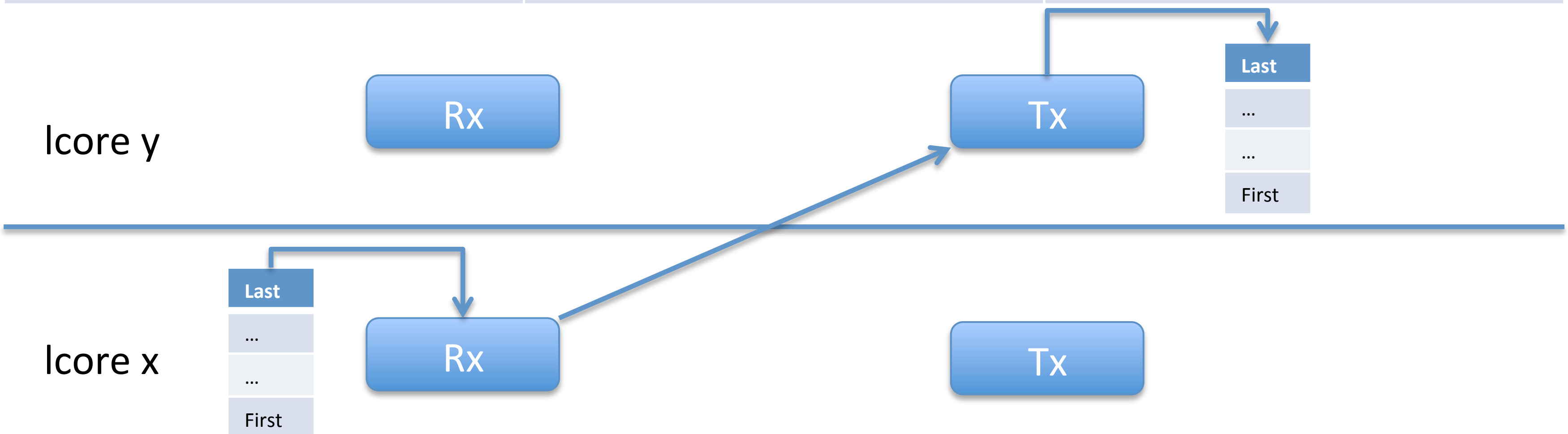
Implementation Observations

DPDK DATA PLANE DEVELOPMENT KIT Userspace 2015

Observation	Description	Solution/Workaround
UIO eliminates the traditional *nix methods of managing devices	The ability of a legacy application to adopt DPDK may be hindered by the lack of /dev/<network device> which precludes the use of ethtool, ifconfig and similar utilities or programmatic access to related ioctls.	Alternatives considered: • KNI • Bifurcated driver • Ethtool/netdevice APIs Ultimately we chose to pursue Ethtool/ Netdevice APIs.
Poll mode 100% core utilization with idle interfaces	It may not be acceptable to consume a core when there is no work.	There is no easy way to solve this problem, but we are hopeful that Rx interrupt support will provide relief.

Implementation Observations

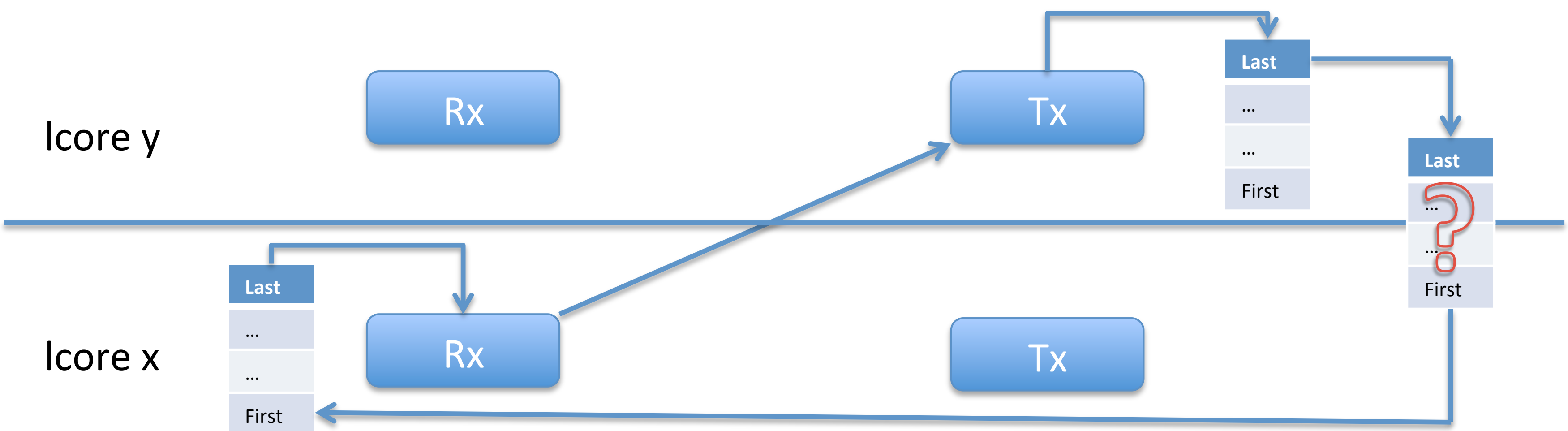
Observations	Description	Solution/Workaround
Per-lcore pktmbuf cache appears to be optimized for rx & tx on the same lcore	Experimented with various models, and observed lcore cache can be underutilized or can become an pktmbuf parking lot. This can occur if rx and tx are on different lcores, if an imbalance exists between rx and tx or an lcore infrequently does rx/tx.	Adapt application. APIs to monitor and manipulate local cache.



Implementation Observations

DPDK DATA PLANE DEVELOPMENT KIT
Userspace 2015

Observations	Description	Solution/Workaround
Altering size of pktmbuf pool affects performance	Tested different pktmbuf pool sizes under various models. Under some conditions, pktmbuf pool increase resulted in reduced performance. We speculate that the global pool algorithm is FIFO-like and a large pool has negative cache consequences.	Design to maximize effectiveness of local pktmbuf cache. Could the pktmbuf pool algorithm be altered to improve performance?



Implementation Observations

Observations	Description	Solution/Workaround
PMDs do not implement all APIs	We began our testing with IXGBE. As we expanded to other NIC/PMDs, we encountered some crashes related to unimplemented APIs. Most but not all FVs are validated.	Be aware that not all PMDs implement all APIs. Consider alternative approaches to FV validation.

Type	DPDK API	PMD FV	Safe	ixgbe	ixgbevf	vmxnet3	virtio	igb	igbvf
eth_dev_ops	rte_eth_dev_tx_queue_stop	tx_queue_stop	✔	✔	✖	✖	✖	✖	✖
eth_dev_ops	rte_eth_rx_queue_seutp	rx_queue_setup	✔	✔	✔	✔	✔	✔	✔
eth_dev_ops	rte_eth_dev_rx_queue_config	rx_queue_release	✔	✔	✔	✔	✔	✔	✔
eth_dev_ops	rte_eth_rx_queue_count	rx_queue_count	✖	✔	✖	✖	✖	✔	✖
eth_dev_ops	NO API	tx_queue_count	✔	✔	✖	✖	✖	✖	✖
eth_dev_ops	rte_eth_rx_descriptor_done	rx_descriptor_done	✖	✔	✖	✖	✖	✔	✖
eth_dev_ops	rte_eth_tx_queue_setup	tx_queue_setup	✔	✔	✔	✔	✔	✔	✔
eth_dev_ops	rte_eth_dev_tx_queue_config	tx_queue_release	✔	✔	✔	✔	✔	✔	✔
eth_dev_ops	rte_eth_led_on	dev_led_on	✔	✔	✖	✖	✖	✔	✖
eth_dev_ops	rte_eth_led_off	dev_led_off	✔	✔	✖	✖	✖	✔	✖
eth_dev_ops	rte_eth_dev_flow_ctrl_get	flow_ctrl_get	✔	✔	✖	✖	✖	✔	✖
eth_dev_ops	rte_eth_dev_flow_ctrl_set	flow_ctrl_set	✔	✔	✖	✖	✖	✔	✖
eth_dev_ops	rte_eth_dev_priority_flow_ctrl_set	priority_flow_ctrl_set	✔	✔	✖	✖	✖	✖	✖
eth_dev_ops	rte_eth_dev_mac_addr_remove	mac_addr_remove	✔	✔	✔	✖	✔	✔	✖
eth_dev_ops	rte_eth_dev_mac_addr_add	mac_addr_add	✔	✔	✔	✖	✔	✔	✖

Implementation Observations

Observations	Description	Solution/Workaround
PMDs are at varying levels of maturity.	PMDs for Intel NICs seem to be most mature. PMDs are not as mature as kernel drivers.	Community awareness
Limited visibility into PMD Capabilities	Not all PMDs support all features/ functions. Some capabilities can be determined at run time from rte_eth_dev_info_get(), some cannot. For example, jumbo is not supported in vmxnet3 today.	Disable feature/function based on PMD driver name. Enumerate and implement methods of querying capabilities.
Limited visibility into PMD Mode	Some PMDs support multiple modes of operation. For example IXGBE supports a vector mode. The decision to use vector mode is made within the PMD but there is no way to programmatically query whether this mode is enabled.	The mode may be printed out, but access to logs may not be an option in a deeply embedded system. Add APIs to query mode.

Implementation Observations

Observations	Description	Solution/Workaround
DPDK may exit(), abort() or panic().	Under some error conditions, DPDK may exit, abort, panic. For example, if there is no memory available on a socket to allocate a memzone for ethernet data, DPDK aborts. It is important to note that the application may still function even though DPDK cannot.	Allow application to set failure policy. Consider prohibiting exit/abort/panic within DPDK. Return errors and allow application to determine appropriate action.
Not all errors are reported to the application	When testing IXGBEVF we discovered that it silently dropped PF NACKs when requesting an MTU change.	Ensure APIs return appropriate indication of success/failure. Ensure that errors are reported up to application level.
There is no mechanism for an application to provide a logging facility	In a deeply embedded application, there may be no mechanism for exposing the output printf to users/administrators.	Use RTE_LOG instead of printf (and friend)s. Applications could then override RTE_LOG. Consider provided a mechanism for the application to bind a logging facility and use RTE_LOG or similar for all logging.

Implementation Observations

Observations	Description	Solution/Workaround
32bit applications are a challenge	Examples: • 64bit KNI kernel module does not support 32bit compatibility IOCTLs • Initial IXGBE vector mode did not support 32bit user space.	Extensive testing and verification. Community awareness that there are 32bit applications. IXGBE 32bit vector mode support was added in DPDK 2.0.
Configuring DPDK within a VM can be a challenge	Examples: • VMs have little visibility into NUMA. • DPDK NUMA logic falling back to physical_device_id • Memory channel configuration is unknown.	NUMA fallback resolved in DPDK 2.0. There is currently no way to determine programmatically memory channel layout from within a VM.
Dynamic linking assumed	Ethernet driver registration is in the library constructor.	Wrap DPDK static libs between –whole-archive and --no-whole-archive
Supporting legacy kernels, tools and libraries can be challenging	Legacy kernels and tools may not support all of the features and functionality expected by DPDK.	Extensive testing and verification by those who require legacy support. Community awareness that there are applications which must support legacy models.

DPDK

Implementation Moving Forward

Implementation Moving Forward

- To date our focus has been integrating DPDK into our application.
- We have worked closely with Intel to resolve the issues and upstream to dpdk.org.
- Going forward our intent is to contribute directly to DPDK through bug fixes and enhancements.

DPDK

Q&A