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

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
Implementation Observations
<table>
<thead>
<tr>
<th>Observation</th>
<th>Description</th>
<th>Solution/Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapting multi-threaded applications with more threads than physical cores can be difficult</td>
<td>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.</td>
<td>From DPDK 2.0 forward, multi-lcore/physical core is supported.</td>
</tr>
<tr>
<td>Multi-socket VM initialization failures on legacy kernels</td>
<td>Configuring multiple virtual sockets in a VM, particularly with legacy kernels, could result in NUMA panic.</td>
<td>Issue addressed in DPDK 2.0 and forward by removing the fall back to the physical_device_id.</td>
</tr>
</tbody>
</table>
### Implementation Observations

<table>
<thead>
<tr>
<th>Observation</th>
<th>Description</th>
<th>Solution/Workaround</th>
</tr>
</thead>
</table>
| 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. |
<p>| 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. |</p>
<table>
<thead>
<tr>
<th>Observations</th>
<th>Description</th>
<th>Solution/Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per-lcore pktmbuf cache appears to be optimized for rx &amp; tx on the same lcore</td>
<td>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.</td>
<td>Adapt application. APIs to monitor and manipulate local cache.</td>
</tr>
</tbody>
</table>

**Diagram:**

- **lcore x**
  - Rx
  - ...
  - ...
  - First

- **lcore y**
  - Rx
  - ...
  - ...
  - Last

- **Tx**
  - Last
  - ...
  - First
### Implementation Observations

<table>
<thead>
<tr>
<th>Observations</th>
<th>Description</th>
<th>Solution/Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altering size of pktmbuf pool affects performance</td>
<td>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.</td>
<td>Design to maximize effectiveness of local pktmbuf cache. Could the pktmbuf pool algorithm be altered to improve performance?</td>
</tr>
</tbody>
</table>

**Diagram:**
- **lcore y**:
  - Rx
  - Tx
- **lcore x**:
  - Rx
  - Tx

The diagram illustrates the flow of data between two cores (lcore y and lcore x) with Rx and Tx indicating receive and transmit operations, respectively. The flow is bidirectional, indicating both receive and transmit operations occur concurrently on each core.
Observations

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.

Solution/Workaround

Be aware that not all PMDs implement all APIs. Consider alternative approaches to FV validation.
### Implementation Observations

<table>
<thead>
<tr>
<th>Observations</th>
<th>Description</th>
<th>Solution/Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDs are at varying levels of maturity.</td>
<td>PMDs for Intel NICs seem to be most mature. PMDs are not as mature as kernel drivers.</td>
<td>Community awareness</td>
</tr>
<tr>
<td>Limited visibility into PMD Capabilities</td>
<td>Not all PMDs support all features/functions. Some capabilities can be determined at run time from <code>rte_eth_dev_info_get()</code>, some cannot. For example, jumbo is not supported in vmxnet3 today.</td>
<td>Disable feature/function based on PMD driver name. Enumerate and implement methods of querying capabilities.</td>
</tr>
<tr>
<td>Limited visibility into PMD Mode</td>
<td>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 programatically query whether this mode is enabled.</td>
<td>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.</td>
</tr>
<tr>
<td>Observations</td>
<td>Description</td>
<td>Solution/Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DPDK may exit(), abort() or panic().</td>
<td>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.</td>
<td>Allow application to set failure policy. Consider prohibiting exit/abort/panic within DPDK. Return errors and allow application to determine appropriate action.</td>
</tr>
<tr>
<td>Not all errors are reported to the application</td>
<td>When testing IXGBEVF we discovered that it silently dropped PF NACKs when requesting an MTU change.</td>
<td>Ensure APIs return appropriate indication of success/failure. Ensure that errors are reported up to application level.</td>
</tr>
<tr>
<td>There is no mechanism for an application to provide a logging facility</td>
<td>In a deeply embedded application, there may be no mechanism for exposing the output printf to users/administrators.</td>
<td>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.</td>
</tr>
<tr>
<td>Observations</td>
<td>Description</td>
<td>Solution/Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>32bit applications are a challenge</td>
<td>Examples:</td>
<td>Extensive testing and verification. Community awareness that there are 32bit applications. IXGE 32bit vector mode support was added in DPDK 2.0.</td>
</tr>
<tr>
<td></td>
<td>• 64bit KNI kernel module does not support 32bit compatibility IOCTLs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Initial IXGBE vector mode did not support 32bit user space.</td>
<td></td>
</tr>
<tr>
<td>Configuring DPDK within a VM can be a challenge</td>
<td>Examples:</td>
<td>NUMA fallback resolved in DPDK 2.0. There is currently no way to determine programatically memory channel layout from within a VM.</td>
</tr>
<tr>
<td></td>
<td>• VMs have little visibility into NUMA.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DPDK NUMA logic falling back to physical_device_id</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Memory channel configuration is unknown.</td>
<td></td>
</tr>
<tr>
<td>Dynamic linking assumed</td>
<td>Ethernet driver registration is in the library constructor.</td>
<td>Wrap DPDK static libs between --whole-archive and --no-whole-archive</td>
</tr>
<tr>
<td>Supporting legacy kernels, tools and libraries can be challenging</td>
<td>Legacy kernels and tools may not support all of the features and functionality expected by DPDK.</td>
<td>Extensive testing and verification by those who require legacy support. Community awareness that there are applications which must support legacy models.</td>
</tr>
</tbody>
</table>
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.