Agenda

- OvS Offload Layer
- OvS Offload Capabilities
- OvS Offload Registration
- OvS-DPDK Offload Thread Model Issues
- Tunnel Decap Design Challenges
- OvS-DPDK Partial Action Offload Design Challenges
- Differences between User Mode and Kernel Mode Offload Data Paths
- Summary
OvS Offload Layer

- Implements control path for flow offloads
- Device agnostic
- Hidden from ofproto layer

- Split in generic and provider sub-layers
- Enables multiple NIC flow offload APIs
- Flow APIs registered by specific provider
- Two subsets of flow APIs:
  - Put, Delete, and Stat APIs for a specific flow
  - Create, Destroy, etc. APIs for flow dumps

Challenges with this layer – Focus of this talk
OvS Offload Capabilities

- Match Fields and Actions supported by a device can’t be expressed
- DPDK provides `rte_flow_validate()`
  - But needs an additional trip to the device for every flow offload
  - OvS currently does not use `rte_flow_validate()`
- Kernel TC Flower does not have equivalent of `rte_flow_validate()`

- **OvS Offload layer can be optimized to support device flow offload capabilities**
  - Could be a simple bitmask of match and actions supported
  - Exported by each offload capable device
  - Offload layer can use this bitmap before offloading to a device
Each provider (e.g. netdev-offload-dpdk) registers a DP specific flow_api object.
Registration is done at the time of offload provider initialization.
flow_api object is added to a global list of registered flow APIs.
At the time of netdev creation, the corresponding flow APIs are initialized.

Issues
- Assumes every eth device supports flow-api
- No device (PMD) specific check
- If ‘hw-offload’ enabled in OVS, offload attempted on any eth_dev attached to OVS
OvS-DPDK Offload Layer Thread Model Issues

- PMD threads process packets, handle DP misses, OF classification
- Offload request is deferred to an offload thread
- Scheduling latency is involved in running the offload thread
- N PMD-threads : 1-offload-thread (serialization across multiple devices)
- Lack of offload error propagation back to PMD threads due to this model
- Lack of infra to share data and synchronize DP and offload threads

Concurrency and Synchronization can Improve Offload Layer Thread Model
Tunnel Decapsulation HW-Offload Sequence

Solid arrows: Pkt sequence
Dotted arrows: Flow sequence
Packet and Flow Processing Sequence Enumerated

- Diagram shows ingress packet/flow sequence
- Solid arrows: packet traversal; dotted arrows: flow processing
- (1) First tunneled packet from the wire received by the PMD (via upink/PF)
- (2) Packet is received by OVS datapath (when OVS polls PF)
- (3) No datapath rule (flow miss); upcall made to classify the packet
- (4) Ofproto classifies the packet; creates a datapath rule with actions
- (5) Datapath rule/action executed
  - tnl_pop() and recirculate the packet to tunnel port
- (6) Packet is received by OVS datapath (in the ctx of VXLAN vPort)
- (7) No datapath rule (flow miss); upcall made to classify the packet
- (8) Ofproto classifies the packet; creates a datapath rule with actions
Packet and Flow Processing Sequence Enumerated

• (9) Datapath rule/action executed (forward); packet sent down to the VF-Rep
• (10) VF-Rep transmits packet down to the PF
• (11) PF loops the packet to the VM via the VF
• (12) Datapath adds the flow; initiates an offload request (F2)
• (13) Offload layer issues a rte_flow_create() to the PMD
• (14) PMD programs HW tables
• Control returns back to datapath in the ctx of the PF
• (15) Datapath adds the flow; initiates an offload request (F1)
• (16) Offload layer issues a rte_flow_create() to the PMD
• (17) PMD programs HW tables
• (18) Next packet from the wire decapsulated and sent directly to VM via VF
Tunnel Decap involves two flows and recirculation in OvS
- Flow-F1 (Match: t_dmac, t_dip, t_proto, t_port; Action: Tunnel pop and Output to tunnel port)
- Flow-F2 (Match: t_dip, t_sip, t_id, inner eth, Action: Output to VF-Rep)

Packet can’t be processed entirely in HW, until both flows are offloaded

Decap Flow Offload Sequences can be different (F2→F1, F1→F2, F2 only)
- PMDs can not assume a specific sequence
- PMDs need to internally handle all possible sequences

Tunnel metadata handling is complex
- OVS SW datapath action is “tnl_pop” for F1, SW DP passes tunnel header as metadata
- HW can’t really pop tunnel header when F1 is offloaded (otherwise it loses tunnel metadata)
- HW miss on F2: Packet couldn’t be decapsulated since there is no F2 in HW (packet hit F1)

Statistics: Double counting of F1 for F2 miss in HW complicates the design

Mapping tunnel vPort to Phy port: otherwise F2 is offloaded on all phy ports
Challenges in extending partial offload infrastructure for action offload

Partial offload currently supports only classification offload: Flow match

Partial Action Offload RFC
- Idea is to extend partial offload to support real actions
- Actions like tunnel-encap/decap, vlan push/pop offloaded to HW
- HW classifies + executes specified actions

Challenges
- Today, partial offload is only supported on the ingress device
- Scenarios that involve a SW ingress, but a HW egress offload are not considered
- Deferred offloading in the context of a separate offload thread creates transient out-of-sync
- PMD threads may continue processing actions after the flow was already offloaded
- Lack of APIs to determine whether a flow is eligible for partial actions offload
- An additional problem with ingress-partial-action is lack of data path assistance

OvS Datapath and Offload Layer Designs are not Amenable to Partial Actions Offload
## User/Kernel DP Offload Differences

<table>
<thead>
<tr>
<th>Kernel-DP/Offload</th>
<th>User-DP/Offload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handler threads process flow-miss/upcall</td>
<td>PMD threads process flow-miss/upcall</td>
</tr>
<tr>
<td>Flow added to either DP or offloaded</td>
<td>Flow always added to DP and offloaded</td>
</tr>
<tr>
<td>Offload attempted first; if fails added to DP</td>
<td>Added to DP first and offload scheduled</td>
</tr>
<tr>
<td>Offload synchronous; handler thread waits</td>
<td>Offload async; dispatched to offload thread</td>
</tr>
<tr>
<td>Offl errors returned to initiating thread</td>
<td>Offl errors not returned to initiating thread</td>
</tr>
<tr>
<td>Dynamic rebalancing supported</td>
<td>Dynamic rebalancing unsupported</td>
</tr>
<tr>
<td>Single (logical) flow table; no duplicate flows</td>
<td>Flow table per-port, per-PMD; offload handles duplicate flow-add</td>
</tr>
</tbody>
</table>

**Inconsistencies between User and Kernel Offloads**
• OvS Offload Layer Design is Complicated

• Offload Capability and Discovery is primitive

• Serialized Threading Model poses challenges for partial actions offload

• Two bridge model poses significant challenges for tunnel decap offload

• Differences between user and kernel mode offloads need to be reconciled

• Overall, redesign of OvS offload layer should be considered