AGENDA

Hardware offloads in userland OVS
Parallel offload architecture
Implementation notes
Results
HARDWARE OFFLOADS IN USERLAND OVS
HARDWARE OFFLOADS IN USERLAND OVS

OVS: multi-layer switch

Increasing processing time

RX

Wildcard support

Hardware Classifier

PMD thread

Miss

Miss

Lookup: hashmap
No wildcard

1 table

EMC

Lookup: TSS
Wildcard support

Datapath Classifier

4 sub-tables

ofproto Classifier

NIC

TX

OVS
HARDWARE OFFLOADS IN USERLAND OVS

OVS: multi-layer switch

Increasing processing time

RX

Wildcard support

Hardware Classifier

PMD thread

Lookup: TSS
Wildcard support

4 sub-tables

Datapath Classifier

ofproto Classifier

Miss

NIC

TX

OVS
HARDWARE OFFLOADS IN USERLAND OVS

OVS: multi-layer switch

PMD thread

Wildcard support

Miss

Sync

NIC

Hardware Classifier

Datapath Classifier

Lookup: TSS Wildcard support

4 sub-tables

ofo_proto Classifier

OVS

 hardware classifier

NIC

 wildcard support

.datapath classifier

lookup: tss wildcard support

4 sub-tables

ofproto classifier

OVS: multi-layer switch
HARDWARE OFFLOADS IN USERLAND OVS
Thread model: DPDK ports

Hw-offload thread

CPU 1
PMD thread
Insert
Datapath Classifier

CPU 2
PMD thread
Insert
Datapath Classifier

CPU N
PMD thread
Insert
Datapath Classifier

Core-wise
Modify / Delete

Revalidator M

0:PF
1:PF
2:VF-rep
3:VF-rep
HARDWARE OFFLOADS IN USERLAND OVS
Thread model: DPDK ports

Hw-offload thread
Mark-to-flow map
Megaflow-to-mark map

Offload queue
Insert / Modify / Delete

CPU 1
PMD thread
Datapath Classifier

CPU 2
PMD thread
Datapath Classifier

CPU N
PMD thread
Datapath Classifier

Core-wise

Revalidator M

0:PF
1:PF
2:VF-rep
3:VF-rep
HARDWARE OFFLOADS IN USERLAND OVS
Thread model: DPDK ports

- 0:PF
- 1:PF
- 2:VF
- 3:VF

Side view:
- Hw-offload thread
- Port lock
- Mark-to-flow map
- Port lock
- Megaflow-to-mark map
- Port lock
- Offload queue
- Insert / Modify / Delete
- Queue lock

Core-wise:
- CPU 1: PMD thread, Datapath Classifier
- CPU 2: PMD thread, Datapath Classifier
- CPU N: PMD thread, Datapath Classifier

- Revalidator M
Parallel offload architecture
PARALLEL OFFLOAD ARCHITECTURE

Relevant metrics

Optimization is done on 2 metrics:

• Classifier sync latency.
  From the moment the DPCLS has been updated until the hardware received the update (A → B).

• Offload queue depth.
  Number of offload updates waiting in the queue.
PARALLEL OFLOAD ARCHITECTURE

Locking changes 1

0:PF
1:PF
2:VF-rep
3:VF-rep

Hw-offload thread
Mark-to-flow map
Megaflow-to-mark map

PMD thread
Datapath Classifier

CPU 1
CPU 2
CPU N

MPSC Offload queue
Insert / Modify / Delete

Hw-offload thread

HW-offload handle table

Datapath port map lock

Core-wise

PMD thread
Datapath Classifier

Revalidator M

Datapath port map lock

port
lock

port
lock

port
lock

port
lock

Megaflow-to-mark map
PARALLEL OFFLOAD ARCHITECTURE

Locking changes 2

Hw-offload thread

Mark-to-flow map

Megaflow-to-mark map

Datapath port map lock

HW-offload handle table

Core-wise

CPU 1

PMD thread

Datapath Classifier

CPU 2

PMD thread

Datapath Classifier

CPU N

PMD thread

Datapath Classifier

MPSC Offload queue

Insert / Modify / Delete

Revalidator M

0:PF

1:PF

2:VF-rep

3:VF-rep
PARALLEL OFFLOAD INFRASTRUCTURE

Locking changes 3

0:PF
1:PF
2:VF-rep
3:VF-rep

HW-offload handle table

Hw-offload thread

Mark-to-flow map

Megaflow-to-mark map

Datapath port map RW-lock: rlock

MPSC Offload queue

Insert / Modify / Delete

Core-wise

CPU 1

PMD thread

Datapath Classifier

CPU 2

PMD thread

Datapath Classifier

CPU N

PMD thread

Datapath Classifier

Revalidator M

OVS
IMPLEMENTATION NOTES

▪ Fast Mark pool
  ▪ Mark allocation uses an ID pool that has pathological behavior with non-sequential ID freeing.
  ▪ This ID pool also scales poorly with additional threads.
  ▪ It cannot be fixed without removing features used by other modules, that mark allocation does not require.

⇒ A new allocator is proposed: ‘id-fpool’. Functionalities are reduced to the essentials. It is faster and scales better.
IMPLEMENTATION NOTES
Faster mark pool: add

Lower is better.
Intel® Xeon® CPU E5-2650 v3 @ 2.30 GHz
IMPLEMENTATION NOTES

Faster mark pool: del

Lower is better.

Intel® Xeon® CPU E5-2650 v3 @ 2.30 GHz
IMPLEMENTATION NOTES

Faster mark pool: mix

Lower is better.

Intel® Xeon® CPU E5-2650 v3 @ 2.30 GHz
IMPLEMENTATION NOTES
Faster mark pool: rand

1M Rnd: Add-rand(del)-add

Lower is better.
Intel® Xeon® CPU E5-2650 v3 @ 2.30 GHz
IMPLEMENTATION NOTES

- Fast Mark pool
  - Mark allocation uses an ID pool that has pathological behavior with non-sequential ID freeing.
  - This ID pool also scales poorly with additional threads.
  - It cannot be fixed without removing features used by other modules, that mark allocation does not require.

  ➔ A new allocator is proposed: ‘id-fpool’. Functionalities are reduced to the essentials. It is faster and scales better.

- MPSC queue
  - Thread model is heterogeneous: Affined (PMD) and non-affined threads are all using the queue.
    - Unfair lock (spinlock) is thus not usable.
    - Fair lock (mutex) does not scale (+ worsen CPU coherency traffic).
  - Multi-Producer, Single-Consumer case is common.

  ➔ A fast MPSC queue would be a useful addition to OvS.
Lower is better.
Intel® Xeon® CPU E5-2650 v3 @ 2.30 GHz
RESULTS
Setup

- Measures made on NVIDIA BlueField-2 DPU cores (Low power ARMv8.1 @ 500MHz).
- 4 PMD threads, 3 revalidators (default).
- Traffic made to trigger continuous updates.
- Latency is measured with an Exponential Moving Average, configured with a factor of 0.019802.
  - It gives the same ‘center of mass’ as a Simple Moving Average with a window of 100 entries.
  - Will respond quicker to recent changes, to show correlation with the offload queue depth.
RESULTS

Baseline queue depth & latency

- Latency (EMA)
- Queue depth
RESULTS

Optimized, 1 thread

seconds

ms

entries

latency (EMA)  Queue depth
RESULTS

Optimized, 1 thread

- **ms**
- **seconds**
- **n entries**

- Orange line: latency (EMA)
- Blue line: Queue depth
POSSIBLE IMPROVEMENTS

- Per-port offload table sharding.

  Divide each port offload table into K smaller tables, for K hw-offload threads. It would remove the last contention in the HW offload management.
PARALLEL OFFLOAD ARCHITECTURE

Per-port offload maps

Hw-offload 1
- M-to-f map
- Mf-to-M map
- MPSC queue

Hw-offload 2
- M-to-f map
- Mf-to-M map
- MPSC queue

Hw-offload K
- M-to-f map
- Mf-to-M map
- MPSC queue

Datapath port map RW-lock: rdlock

Core-wise
- CPU 1: PMD thread
- CPU 2: PMD thread
- CPU N: PMD thread

PMD thread
- Datapath Classifier

Scalable Mark pool

Hash(UFID) % K

Port-wise
- 0:PF
- 1:PF
- 2:VF-rep
- 3:VF-rep

HW-offload handle table
- Lock

Datapath port map RW-lock: rdlock

OVS
POSSIBLE IMPROVEMENTS

- Per-port offload table sharding.
  
  Divide each port offload table into K smaller tables, for K hw-offload threads. It would remove the last contention in the HW offload management.

- Improve memory efficiency.
  
  Offloads data structure have been reorganized to allow parallel disjoint access. Beyond parallelism, memory access could be more cache-conscious:
  
  - spatially: reduce match footprint in offloads by avoiding a full description. Alternatively, offload match could be directly written by offload initiator (PMD / revalidator).
  
  - temporally: batch offload updates.