High Frequency Trading and other Workloads on Advanced NICs

Anjali Singhai Jain, Intel Corporation

Priyalee Kushwaha, Intel Corporation

Vadivel Kannappa, Intel Corporation

Joshua Hay, Intel Corporation

Magnus, Karlsson, Intel corporation

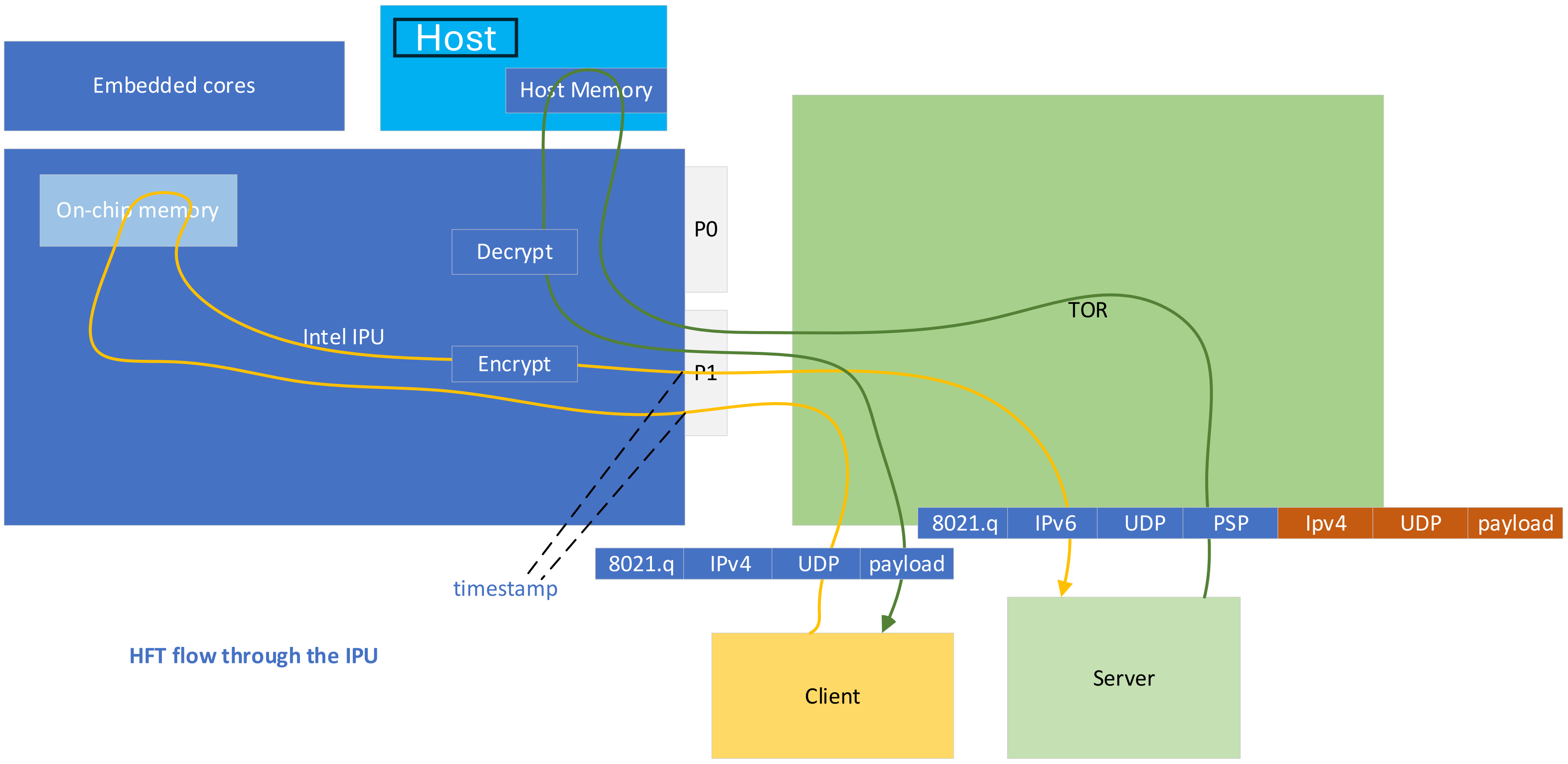
Agenda

- HFT Requirements
- 3 Use cases
  - Low latency, low jitter dedicated TC in HW with other offloads for HFT
  - Per flow Fixed Delay
  - EDT (Earliest Departure Time) using Fair Queue Scheduling offload onto the NIC

1. HFT (High frequency trading) Workload NIC requirements

- Very low Latency
- Very little deviation in latency per packet per flow (Very tight tail latency, low jitter)
- Low latency and High Priority Traffic class in Device to separate the HFT flows from rest of the flows
- Time synchronization between system and Network Device (PTM and PTP)
- Flow Identification and flow scheduling/fixed delay etc (EDT offload on devices through linux kernel.)
- Inline Crypto offload
- Multicast replication

HFT: Low latency, High Priority TC



HFT flow through the IPU

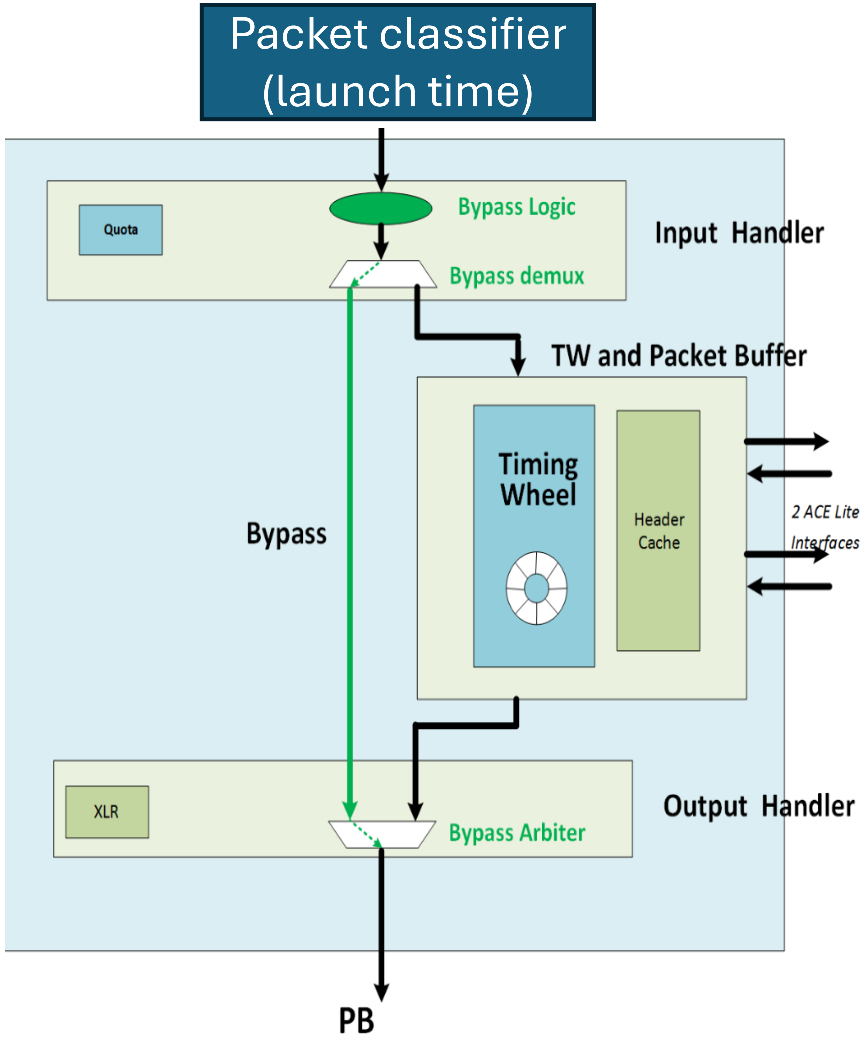
HFT flow through the NIC

- Flows to be identified belonging to low latency TC and then a different ordering domain selected for the low latency flows so that they are not Head of line blocked by the regular flows.
- HW Port 2 Port queue support
- Queue and the buffers for low latency are mapped on OCM (ARM) and so they avoid PCIE latency and jitter.
- PTP helps in making sure, the packets get timestamped at ingress and egress to monitor the packet latency needed for user logs.

2. Per flow fixed delay HW offload

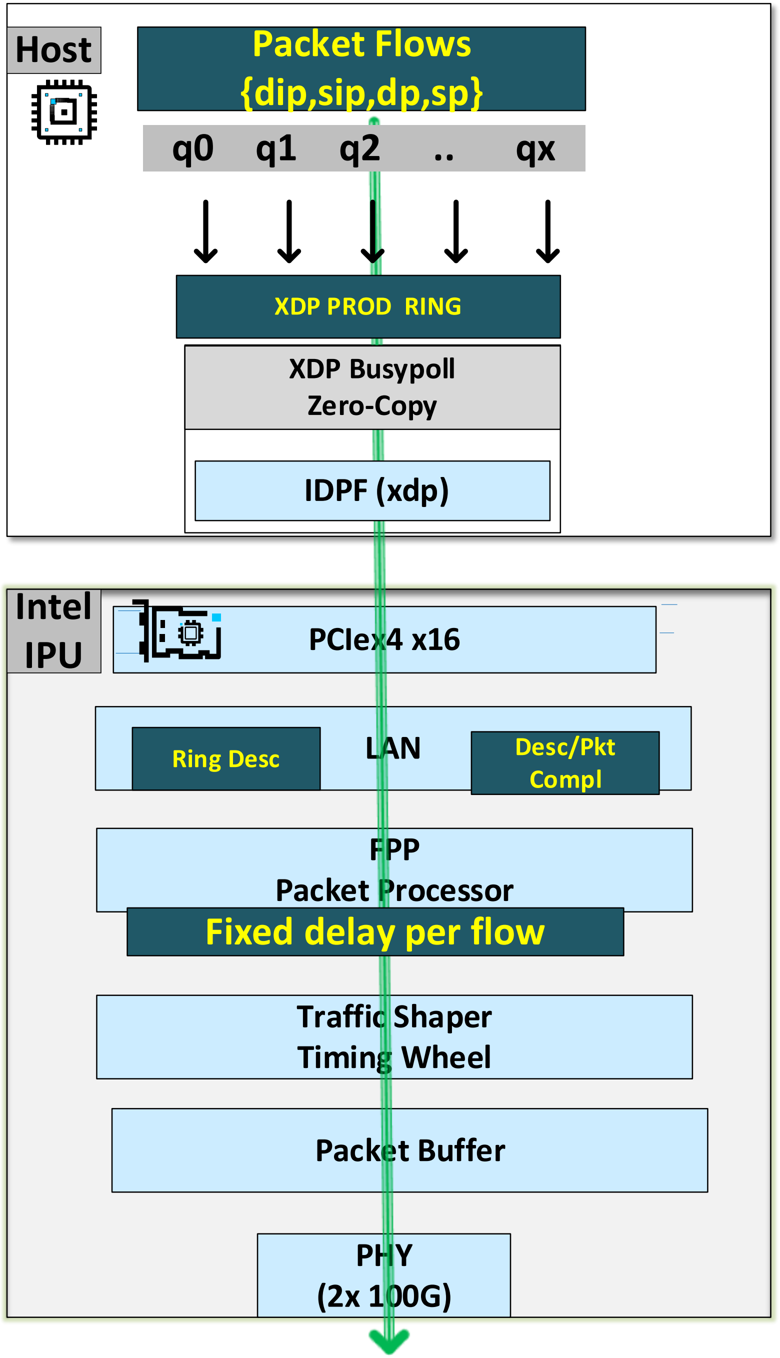
- Helps with scaling to large number of flows and achieving line rate, requires the device to handle large number of inflight packets waiting to be scheduled.
- Flow identification
- Packet timestamping and delay add using ALU operations
- Packet delays achieved using Timing Wheel in HW

Timing Wheel : HW Design for flow pacing



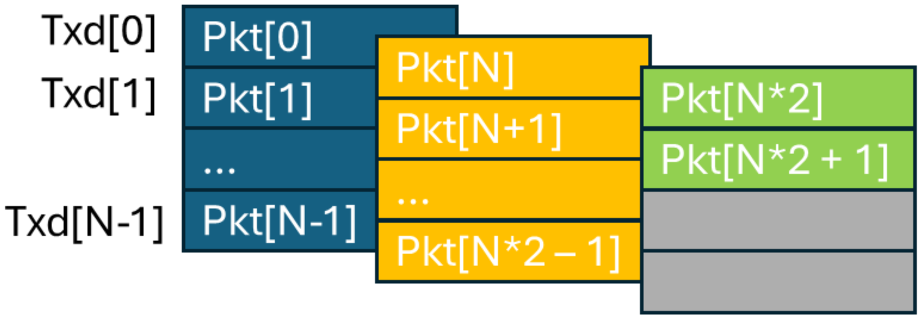
Fixed delay packet flow

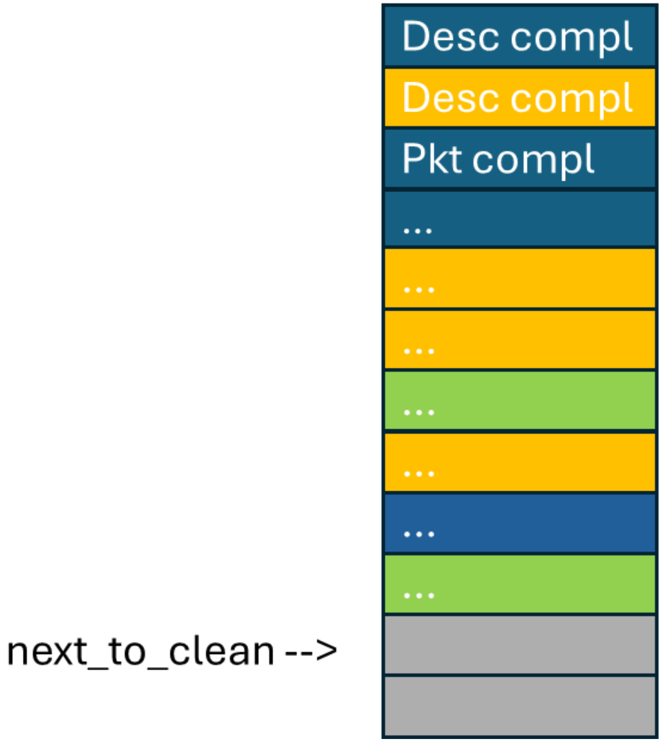
- Flexible packet processor identifies packet flow and adds fixed delay per packet flow.
- Packet shaper timing wheel schedules and buffers delayed packets in HW.
- AF XDP busy-poll with zero copy applications.



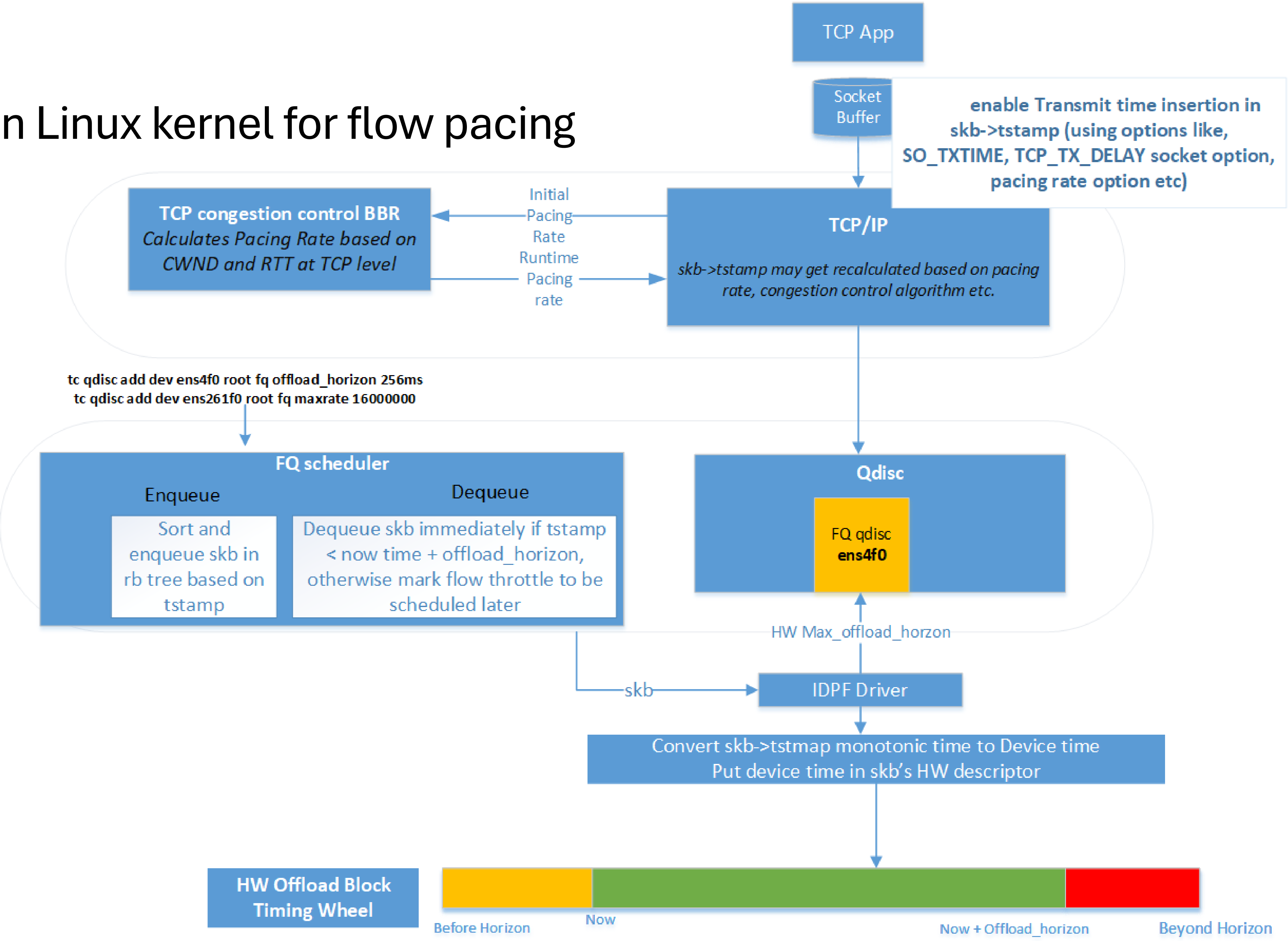
Enabling More than TxQ[size] AF_XDP Packets in Flight

- Completion queue is used to process descriptor completions and buffer completions
  - Descriptor completions tell driver that HW is done reading the descriptors
  - Packet completions tell driver the HW has DMA'ed the packet data (essentially when the packet goes on the wire)
- With this decoupling, driver can reuse the Tx queue descriptors to send more packets while waiting for the paced packet buffer completions
  - Number of completion queue descriptors dictates how many times TxQ can be reused before TX is halted
    - Track pending completions and stop TxQ if there are too many in flight
- Packet completions can arrive in any order (e.g. if different flows are paced at different intervals)
  - Use new API to mark individual XSK frames as completed
    `xsk_tx_complete_ooo(struct xsk_buff_pool *pool, u64 addr)`
- To look for Descriptor completion when posting new tx packets
  `bool xsk_tx_peek_desc_ooo(struct xsk_buff_pool *pool, struct xdp_desc *desc);`





3. EDT enabling in Linux kernel for flow pacing



EDT enabling in Linux kernel for flow pacing

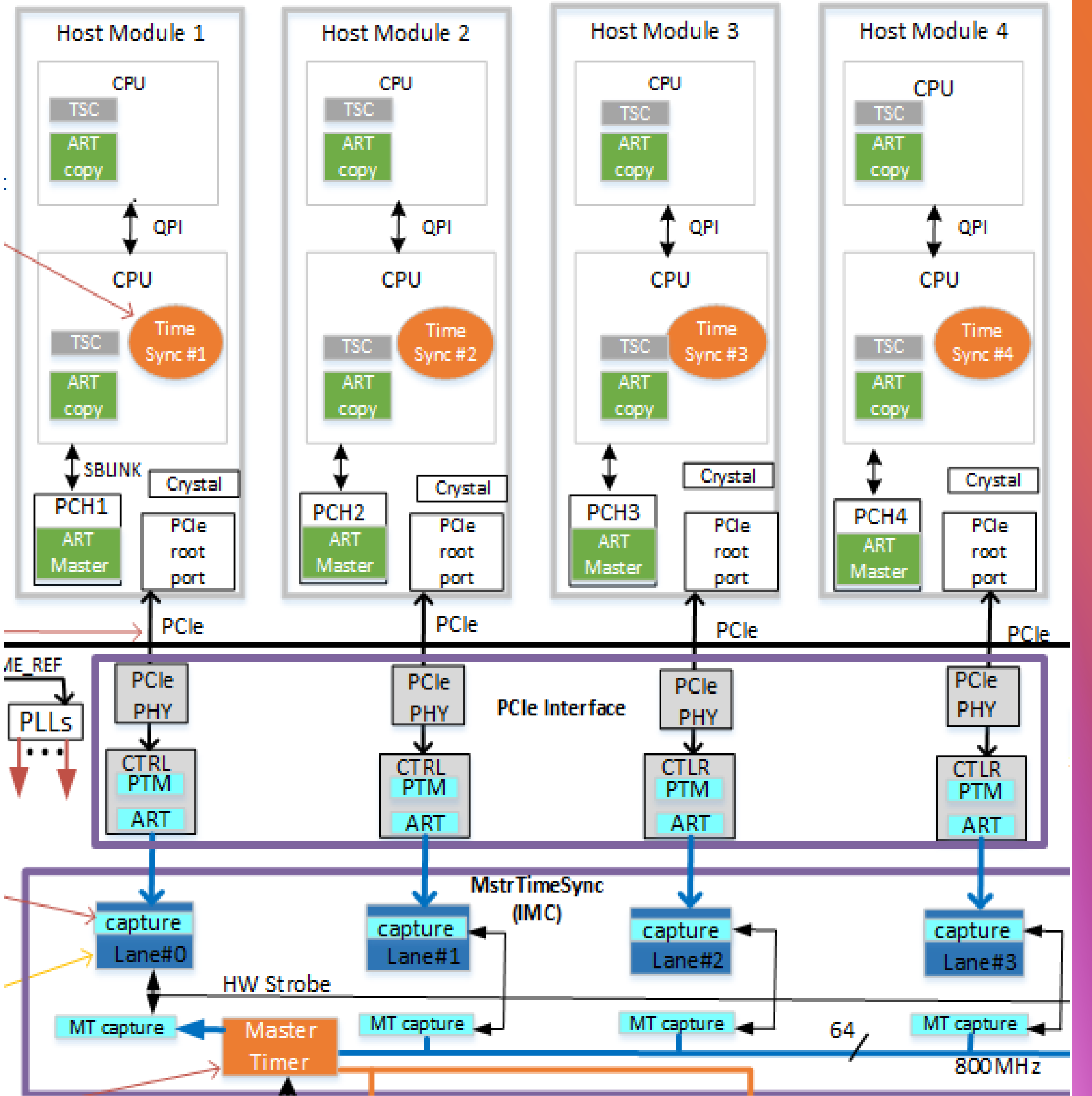
- FQ Qdisc TC Offload (Partial changes are already in kernel)
- IDPF Driver exposes HW capability Maximum offload Horizon to FQ qdisc
- Various ways to insert timestamp in skb, congestion control algorithm, pacing rate, SO_MAX_PACING_RATE and TCP_TX_DELAY socket options
- FQ qdisc dequeues all packets that fits within offload horizon window and rest are throttled to be dequeued later.
- IDPF driver converts skb->tstamp monotonic time to device time. Device time is put into skb descriptor to be used by Timing Wheel HW block
- TCP, FQ and TCP congestion algorithm uses monotonic timer clock for timestamp instead of Real timer clock
- TBD: PTP hooks are used to sync Device timer and Real timer. In future, instead of converting monotonic time to device time (very expensive), a handler will be added in IDPF driver to convert monotonic time to real time, assuming PTP crosstimestamping, PTM etc has synced device and real clock
- PTM PCIe capability provides higher accuracy and low latency for time synchronization.

Offload benefits

- Lower CPU utilization or
- Higher Throughput

PTM (PCIe Precision Time Measurement protocol)

- PTM is periodically initiated by HW
- Host TSC/ ART copy is sent over PCIe
- ART copy and Device time are captured together upon PCIe strobe. That results in great accuracy of both Timer's snapshot at a moment
- PTP SW stack in absence of PTM does not show accuracy, as taking Device time and System time cannot be done in same snapshot.
- Linux PTP Cross-timestamp framework is used to capture ART copy and device timer. Using delta b/w ART copy and device time, Real time clock can be adjusted.
- Prior to PTM, SW mechanisms were used, and the accuracy was in the order of ~1 microsec vs with PTM its ~400 nanoseconds.



Multicast replication

Mirror all incoming multicast packets on eth0 to eth1 interface

```
# tc qdisc add dev eth0 handle ffff: clsact
# tc filter add dev eth0 ingress protocol ip  prior 1\
        flower src_mac 49:aa:bb:cc:dd:ee  \
        action mirred egress mirror dev eth1
```

https://man7.org/linux/man-pages/man8/tc-mirred.8.html

Acknowledgements

- Milena Olech
- Chihjen Chang
- Robert Hathaway
- Naren Mididdadi