Netdev 0x16

Application Device Queues – An Update

Sridhar Samudrala
Amritha Nambiar
Sudheer Mogilappagari

What is ADQ

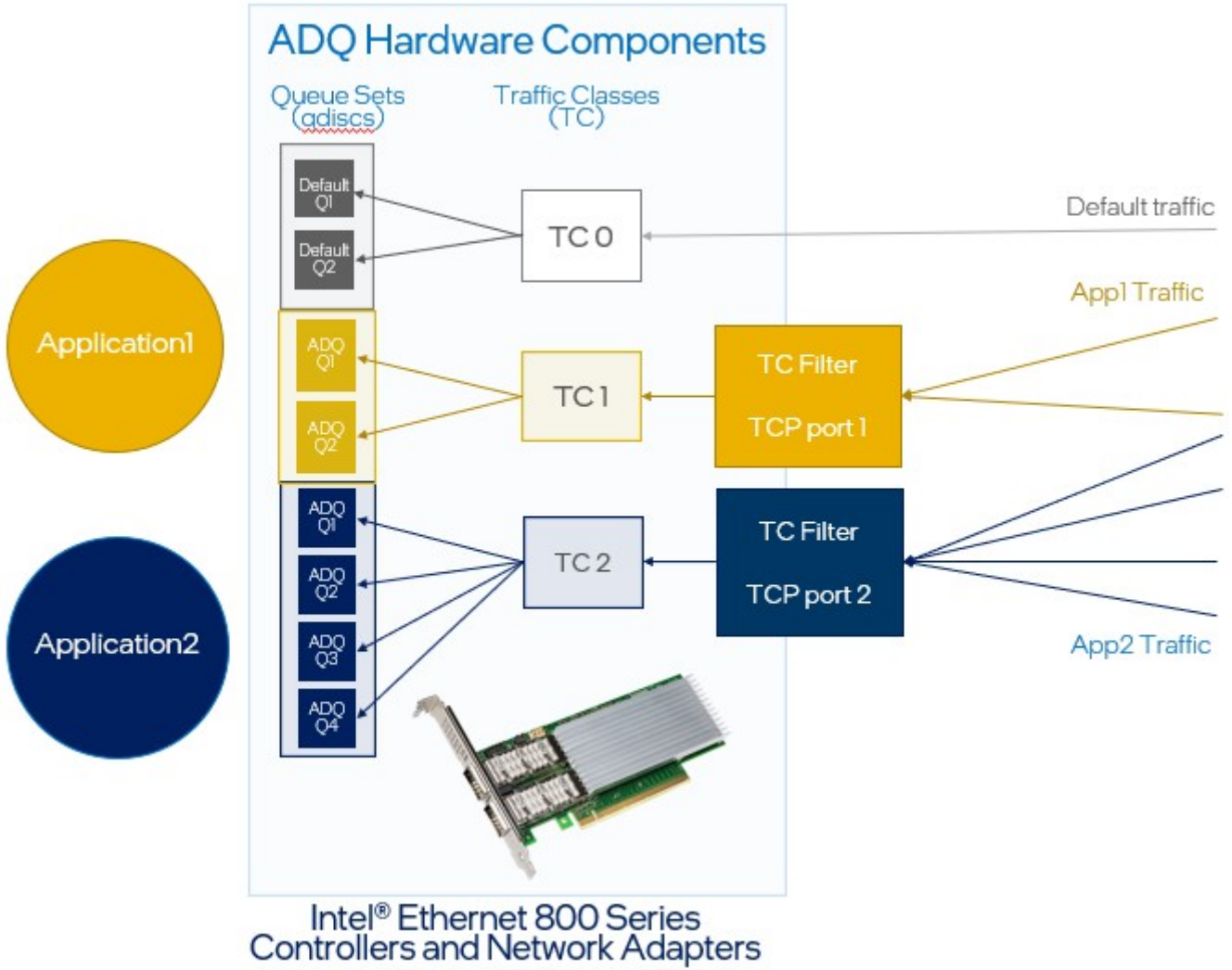
- ADQ is a technology designed to accelerate applications to deliver the predictability and performance required for high-priority, networked workloads.
- ADQ Features (Upstream)
  - Split device queues into multiple dedicated/isolated queue groups per application
  - HW receive filters to direct application traffic to a queue group
  - HW based RSS/Flow Director RX filters to load balance across queues in a queue group
  - SW based TX filters to direct to a queue group/queue
  - Symmetric Queues (XPS based on RXqs) to align TX queue of a flow to the corresponding RX queue
  - TX rate limiting per queue group in HW
  - Application Triggered Busy Polling (busy_poll/napi_deferred_irq/gro_flush_timeout)

How Does ADQ Improve Performance?

1. Utilizes silicon features to steer, load balance and isolate incoming application traffic to dedicated set of hardware queues
2. Allows for a set of queues to have QoS (e.g., rate limits) applied to them
3. Optimizes how data polling is performed

Polling options:

- Application Dependent polling: Provides hints to application for each application thread to service incoming and outgoing traffic from a single device queue
- Application Independent polling



ADQ Queue Groups Configuration

- Device queues split into queue groups backed by HW VSIs
- TC MQPRIO Qdisc

```
tc qdisc add dev <iface> root mqprio
        num_tc 4                          // 4 traffic classes OR queue groups
        map 0 1 2 3                       // priority to TC mapping for TX
        queues 2@0 4@2 8@6 16@14          // 4 queue groups – 2,4,8,16 queues
        hw 1 mode channel                 // offload to HW -  ADQ
```

- RSS context is created per queue group
- Each application thread associated with a single queue in the queue group
  - User load balancing: SO_INCOMING_NAPI_ID socket option (memcached)
  - Kernel load balancing: SO_REUSEPORT_ATTACH_CBPF (nginx)

ADQ Filter Configuration : RX

- Direct incoming traffic to a specific queue in a queue group
- Queue Group Selection

  tc qdisc add dev ens4f0 clsact

  tc filter add dev ens4f0 ingress prio 1 protocol ip flower

  dst_ip 192.168.66.16/32

  ip_proto tcp dst_port 5001

  skip_sw

  hw_tc 1

- Queue Selection
  - RSS within Queue Group
  - ethtool ntuple filters/aRFS

ADQ Filter Configuration : TX

- Direct outgoing traffic to a specific queue in a queue group
- Queue Group Selection
  - Cgroups/SO_PRIORITY socket option
  - TC skbedit – set priority

    tc filter add dev ens4f0 egress prio 1 protocol ip flower
    src_ip 192.168.66.16/32
    ip_proto tcp src_port 5001 action skbedit priority 1
- Queue Selection
  - XPS based on RXq
    - Selects TX queue based on RX queue within the queue group
  - TC skbedit – set queue_mapping

    tc filter add dev ens4f0 egress prio 1 protocol ip flower
    src_ip 192.168.66.16/32
    ip_proto tcp src_port 5001 action skbedit priority 1 pipe
    action skbedit queue_mapping 4

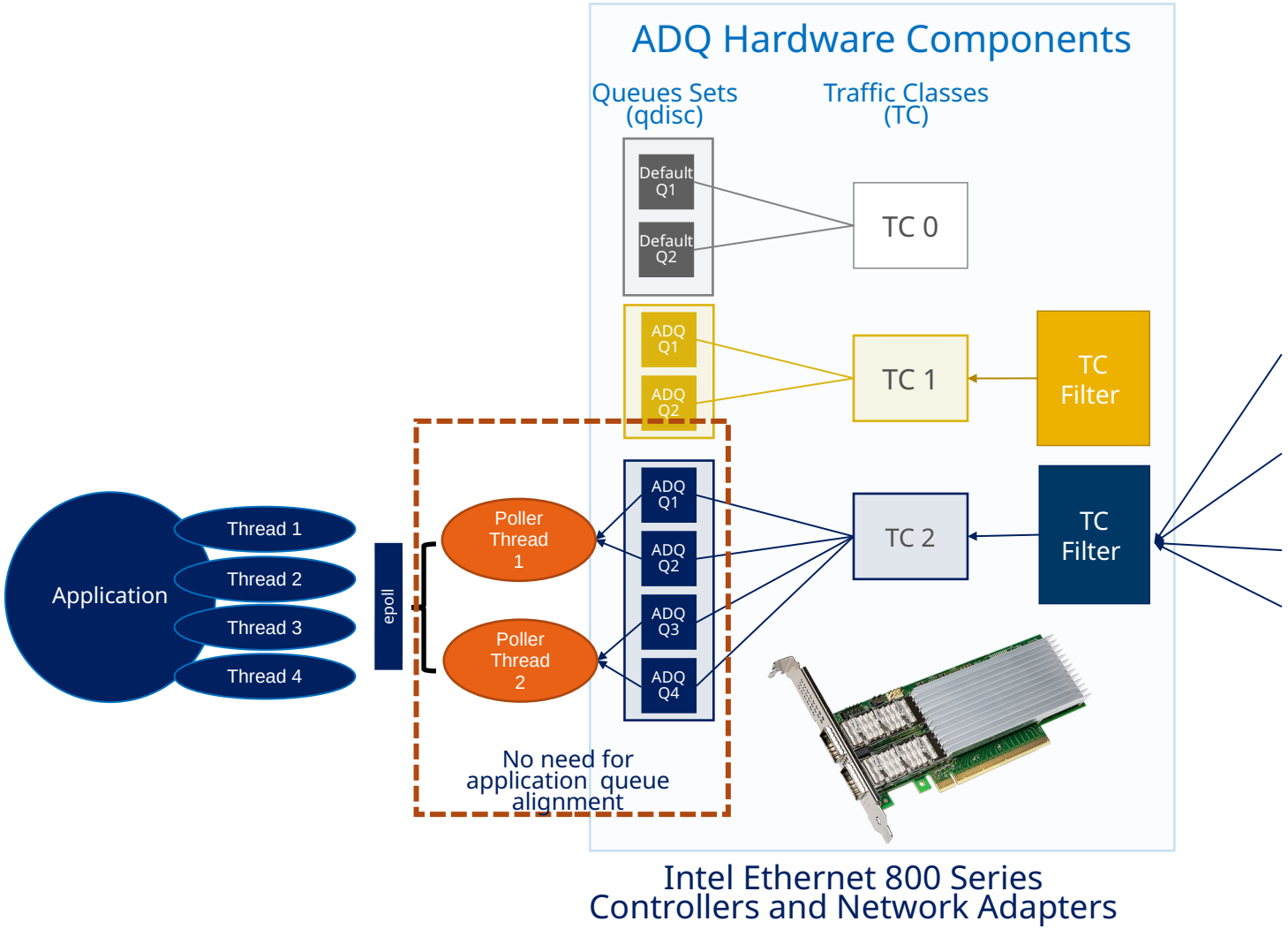
ADQ – New Features

- Application Independent Busy Polling (Independent Pollers)
- HW offloaded RX filters to redirect to an ingress queue using TC ‘skbedit queue_mapping’ action
- Inline Flow Steering to spread flows evenly among the queues in a queue group

ADQ with Independent pollers - Usecases

- Apps that cannot be modified
  - Application dependent Polling requires app thread alignment to device queue to enable app triggered busy polling
- Apps running in K8s Pods
  - Device queue id information is scrubbed before the packet is delivered to the pod namespace
  - veth as the pod network interface
- Apps running in KVM based VMs
  - VM OS is different from the Host OS
    - Host device queue is not accessible from the app running in the VM
  - Virtio-net as the VM network interface

Application Independent Polling



Independent pollers build on standard NAPI polling to allow selected ADQ queues to maintain polling for a longer period.

These independent poller threads poll NIC device queues and push packets to the application socket queues, removing the need to modify application interfaces (ex: epoll) to poll device queues directly.

Resource grouping/aggregation can be done to optimize number of polling CPU cores.

Enables many more environments/use cases (no application change needed).

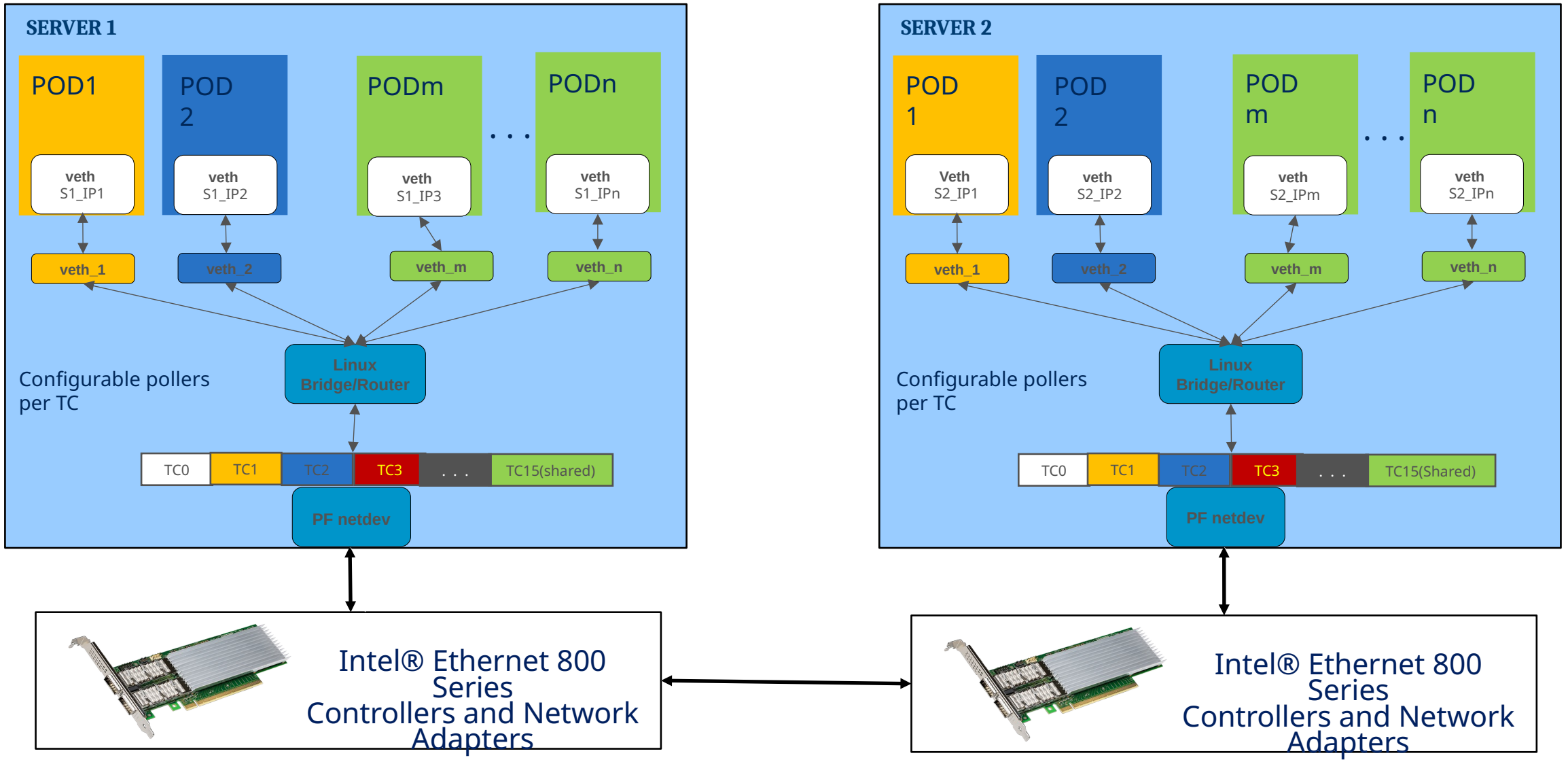
Per TC configurable parameters:

- Number of pollers
- Poller Timeout

ADQ with Independent Pollers - Configuration

- 2 new per rss-context parameters
  - num_pollers
    - Number of pollers per rss-context
    - *ethtool -X <iface> context <context_num> num_pollers <num_pollers>*
  - poller_timeout
    - Timeout in jiffies
    - *ethtool -X <iface> context <context_num> poller_timeout <poller_timeout>*
- By default ‘ksoftirqd’ threads act as independent pollers
- Device specific napi threads
  - Linux kernel 5.13 or newer
  - echo 1 > /sys/class/net/<iface>/threaded
  - Enables pinning of poller threads

K8s VETH Datapath – Accelerated with ADQ



K8s VETH Datapath : Queue Groups

- VETH based K8s Pod network interfaces
- Each ADQ queue group associated with a veth netdev
- Upto 15 exclusive multi queue Pods via RX/TX filters directing to an exclusive queue group
- Much higher number of single queue Pods via RX/TX filters directing to a Queue in a shared Queue Group

```
tc qdisc add dev <iface> root mqprio
        num_tc 16
        map 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
        queues 2@0                                  //    2   default queues
               2@2 2@4 2@6 2@8                      //    4   2-queue pods
               4@10 4@14 4@18 4@22                  //    4   4-queue pods
               8@26 8@34 8@42 8@50                  //    4   8-queue pods
               16@58                                //    1   16-queue pods
               32@74                                //    1   32-queue pods
               150@106                              //  150   1-queue pods
        hw 1 mode channel
```

K8s VETH Datapath : Filters

- CLSACT Qdisc
  - *tc qdisc add dev <iface> clsact*
- Ingress
  - EXCLUSIVE : Forward to TC
    - *tc filter add dev <iface> ingress prio 1 protocol ip flower dst_ip <pod_ip> skip_sw hw_tc <tc_num>*
    - *ethtool -X <iface> context <context_num> inline_fd <on/off>*
  - SHARED : Forward to Queue
    - *tc filter add dev <iface> ingress prio 1 protocol ip flower dst_ip <pod_ip> skip_sw action skbedit queue_mapping <q_num>*
- Egress
  - EXCLUSIVE : skbedit – priority
    - *tc filter add dev <iface> egress prio 1 protocol ip flower src_ip <pod_ip> action skbedit priority <tc_num>*
  - SHARED : skbedit – priority, queue_mapping
    - *tc filter add dev <iface> egress prio 1 protocol ip flower src_ip <pod_ip> action skbedit priority <tc_num> pipe action skbedit queue_mapping <q_num>*

K8s VETH Datapath: Pollers

- Number of Pollers per Queue Group
  - ethtool -X <iface> context 1 num_pollers 1
  - ethtool -X <iface> context 2 num_pollers 1

    ....
  - ethtool -X <iface> context 15 num_pollers 10
- Poller timeout per Queue Group
  - ethtool -X <iface> context 1 poller_timeout 1000
  - ethtool -X <iface> context 2 poller_timeout 1000

    ....
  - ethtool -X <iface> context 15 poller_timeout 1000