# PTQ

***P**er **T**hread **Q**ueues*

Tom Herbert

Linux Plumbers Conference, Aug. 2020

# Abstract

*Assign threads their own dedicated network hardware queues for isolation and performance*

Design areas

- Configuration and management- cgroups
- Transmit - extend XPS
- Receive - extend aRFS
- RX queue processing - interrupt, bust poll, comp queue + sleeping poll

# Assigning queues to threads

- Global queues
  - Abstract out HW queues to global system space
  - *<gqid> -> <device, queue#>*
  - /sys/class/net/<dev>/queues/{rx,tx}-<n>/global_queue_mapping
- New cgroup controller: *net_queues*
  - Network queues are managed as another system resource (
  - Assign pool of “global queues” for transmit and receive.
- Assignment to threads
  - Struct task augmented to contain 16 bit RX  and TX queue
  - At task creation (e.g. fork), assign queues to task
  - Select an RX and TX from cgroup queue ranges

# Transmit path

Transmit path selects using queue assigned to the running thread.

- At socket operations: query *current* task and save TX queues assigned to task in the socket
- On transmit (*netdev_pick_tx*)
  - Check if global TX queue is set in socket
  - Map the global queue to HW queue for TX device
  - Send to resulting queue

# Recive path: Start with aRFS

**aRFS**: Host programs hardware for flows using *ndo_flow_steer* bases on queue->CPU assoc.

aRFS deficiencies:

- Scaling: need to manage state for each flow
- State thrashing: on thread reschedule all reprogram all the threads sockets to different queue for new CPU
- Lag: can only program flow after receiving packets
- OOO packets: when flow switches queues OOO delivery
- No isolation: queues are shared

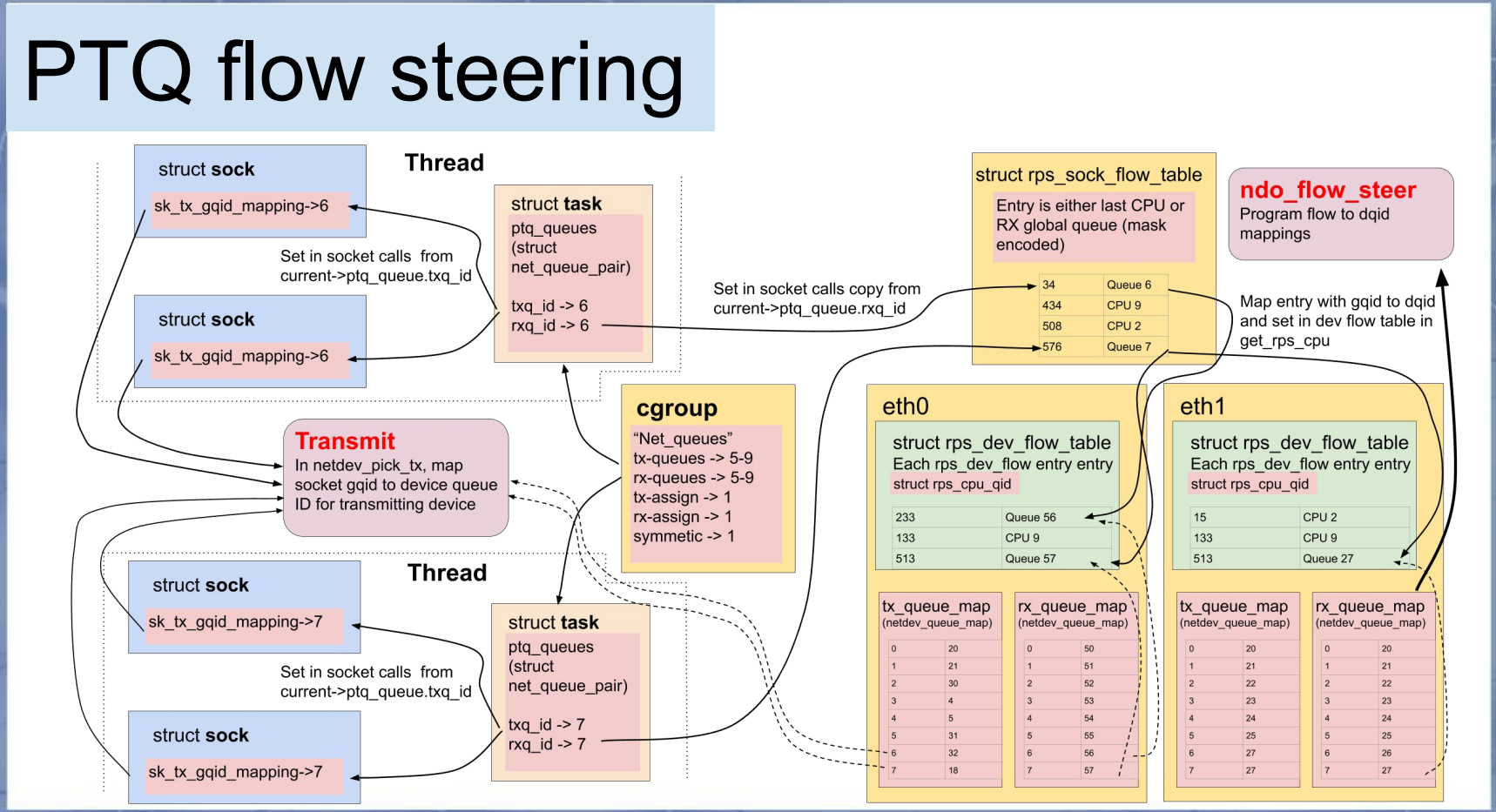
# Extending aRFS

- RFS tables store either CPU or global queue
  - Save queue in socket calls
  - RX queue in sock to RFS table entry
- Do RFS lookup on receive. If global queue is returned
  - Map global queue to HW queue for RX device
  - Check is RX queue equals found HW queue
  - *ndo_flow_steer* on flow to HW queue if different
- Addresses
  - State thrashing: Receive queue follows thread
  - OOO and lag since queue change is now a rare event

# tc flower steering+aRFS in concert

- *hw_tc* can map to 16 queue groups (could extend tc flower to allow “rx_queue <global queue list>”)
- Filter corresponds to application and queue list corresponds to RX queues for the apps cgroup
- **As long as stateless mechanisms give the right answer, don’t need flow state**
- i.e. don’t instantiate HW flow if packets received on right queue, fallback to ndo_flow_steer when wrong
- Addresses isolation+scaling

# PTQ flow steering



# Kicking RX processing

- *Interrupts*: probably not very practical to continuously move interrupt affinity when thread moves
- *Busy polling*: straightforward, but we incur costs of it
- *Sleeping poll with completion queue*
  - Application polls device and sleeps
  - One CPU busy polls on RX completion
  - Reverse map ready RX queue to thread
  - Schedule thread (combine with Shenango for fast scheduling)

# Status

- RFC patch sent to netdev
- Testing and performance analysis started
- Patch was cgroup v1, v2 support seems straightforward

# Futures

- Support for TX prio (per thread TX priority queues)
- Extend hw_tc to allow more than 16 classes. Alternative might be arbitrary group of queues
- eBPF for configurable policies (queue select)
- tc queue steering works great if the #queues == #app_threads, mismatch causes an imbalance
- How to go from stateful full back to stateless?
- RFS tables are caches, can we eliminate them?
- Align SO_REUSEPORT with tc steering
- 64 bit hashes!

End slide