Qdisc layer

Fast enough for 10G wirespeed?

Jesper Dangaard Brouer
Hannes Frederic Sowa
Daniel Borkmann
Florian Westphal

Network-Services-Team, Red Hat inc.

Netfilter Workshop, July 2014

Overview

- Analysing qdisc code path
  - Estimating and measuring overhead
- Batching
  - Results with bulk dequeue
- Lockless FIFO implementation
  - Results and comparisons

Basic question?

- Basic problem/question:
  - Is qdisc code path fast enough for 10G wirespeed?

- 10Gbit/s wirespeed smallest frame:
  - 14.88 Mpps -> Time Budget: **67.2 ns per packet**

Locking: in qdisc code path

- Optimal "fast-path" where the qdisc is empty
- The following locking occurs:
  - In __dev_xmit_skb(): **(1)** spin_lock(root_lock)
    - (detect qdisc empty, allow direct xmit) calling sch_direct_xmit()
  - In sch_direct_xmit(): **(2)** spin_unlock(root_lock)
  - In sch_direct_xmit(): **(3)** HARD_TX_LOCK(dev, txq, smp_processor_id());
  - In sch_direct_xmit(): **(4)** HARD_TX_UNLOCK(dev, txq);
  - In sch_direct_xmit(): **(5)** spin_lock(root_lock);
  - Return to __dev_xmit_skb(): **(6)** spin_unlock(root_lock);
- LOCK cost on this arch: approx 8 ns
  - 8 ns * 6 LOCK-ops = **48 ns pure lock overhead**

Test: Remove qdisc from device

- Remove qdisc from device, like software/virtual devices
  - qdisc path skipped, if (qdisc->enqueue == NULL)
  - Run time hack (by Eric Dumazet)

```
ifconfig $DEV txqueuelen 0
tc qdisc add dev $DEV root sfq
tc qdisc del dev $DEV root
```

- Gotcha's
  - Still takes the TXQ lock `HARD_TX_{LOCK,UNLOCK}`
    - Thus, "only" avoiding 4 LOCK calls on fast-path
  - Don't respect TXQ/device "full" state
    - e.g. ignoring netif_xmit_frozen_or_stopped(txq)

Result: Remove qdisc from device

- Measurement done on CPU E5-2695(ES)
  - with trafgen, packet config: udp_example01
- Using qdisc NULL hack (Clean kernel 3.15.0-rc6)
  - 1566918 pps = 638.20 ns -- qdisc-path
  - 1731000 pps = 577.70 ns -- qdisc NULL hack
  - +164082 pps = **-60.50 ns** -- improvement
- Theory: 4 locks saved = 32ns
  - remaining 2 locks = 16ns
  - Derived remaining cost of qdisc path = 12.5 ns
- **Locking: Significant part of qdisc cost**

Measured overhead vs. wirespeed

- Qdisc code path measured overhead: **60 ns**
  - Problematic for wirespeed **67.2 ns** budget

- How can we reduce this overhead?
  - Batching to amortize locking cost
  - Lockless FIFO implementation

Difficult testing batching

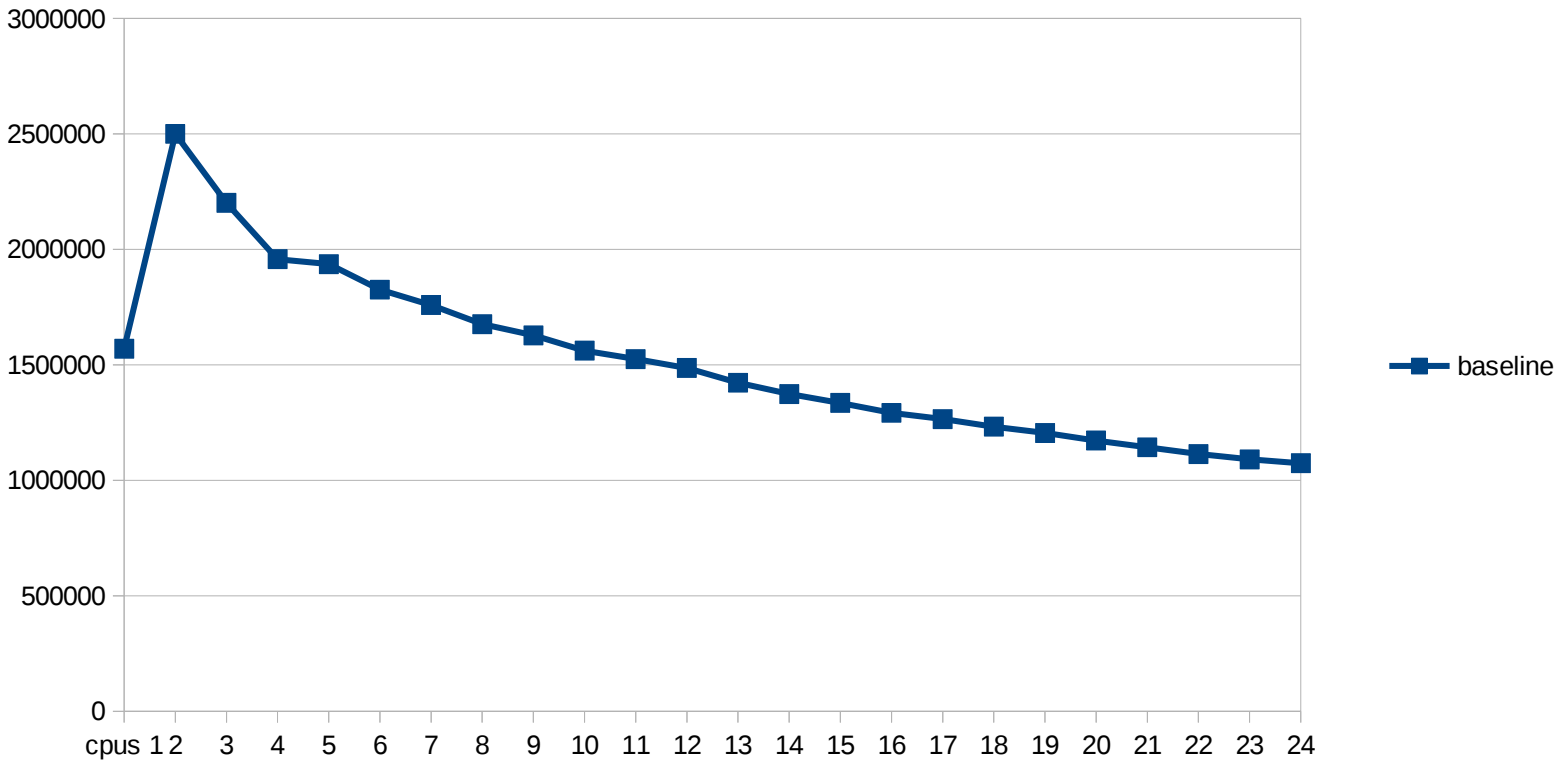
- Must accumulate queue in qdisc
  - Before batching of packets can happen
- Missing fast enough packet generator
  - For single CPU overloading test
    - Because default qdisc tied to HW TXQ queue
- Options
  - External pktgen forwarding traffic
  - Modify pktgen to send to qdisc layer
  - Simply install single queue `pfifo` qdisc

Test setup

- Choose to simply use `pfifo` qdisc
  - Mostly as quick solution
  - Using trafgen, on multiple CPUs (option –cpus)
    - Remember use --qdisc-path option
    - Trafgen packet config:udp_example01.trafgen
      - https://github.com/netoptimizer/network-testing/blob/master/trafgen/udp_example01.trafgen
- Measurements done on CPU E5-2695(ES)

Results: pfifo baseline

- Baseline results, clean kernel 3.15.0-rc6
  - Cpu 1: 1,570,083 pps

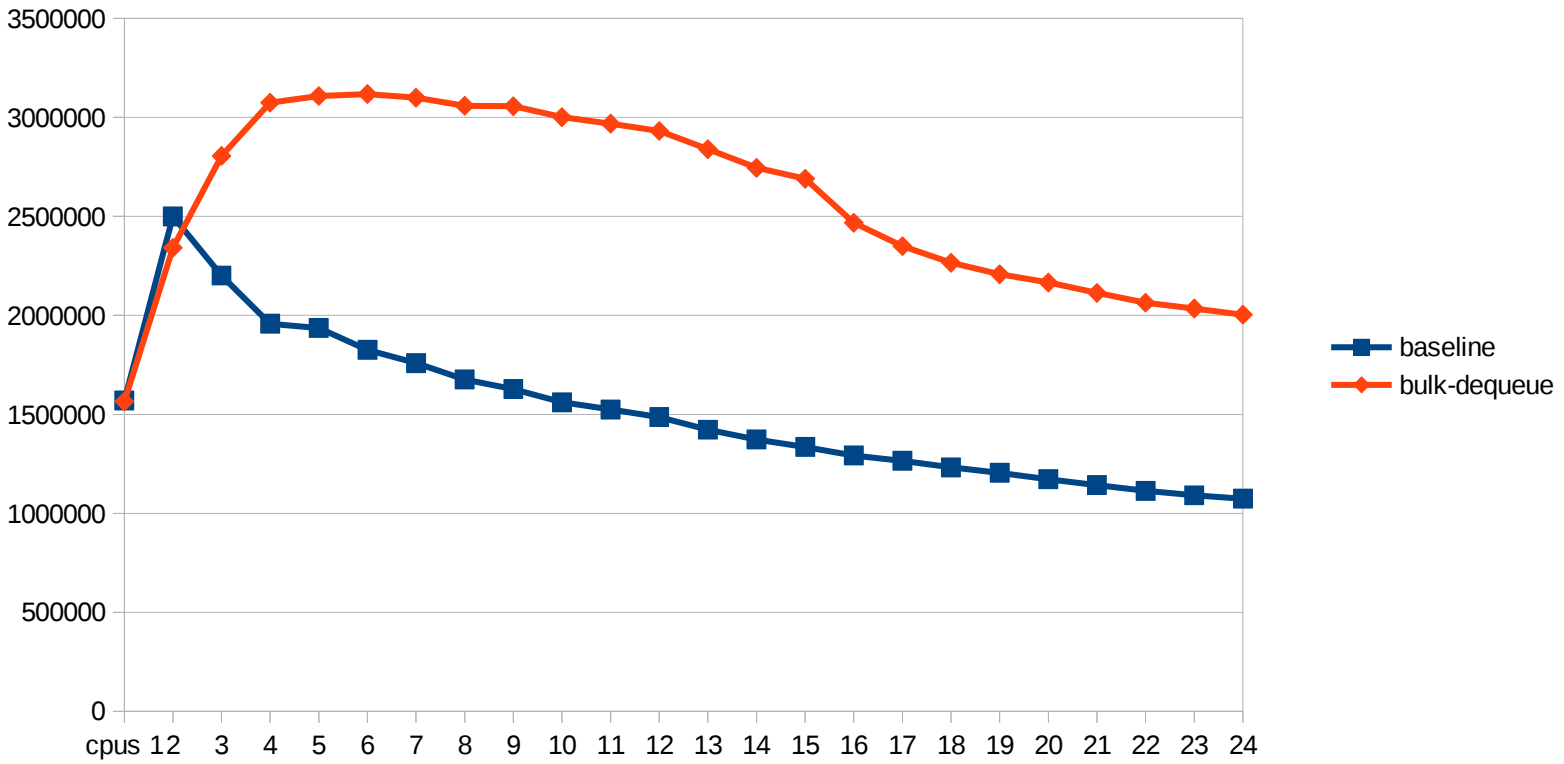


Implement: Bulk qdisc dequeue

- By bulk qdisc ***dequeue***
  - Amortize root lock, unlock+lock per N packets
    - Actually helps enqueue'ers “get” root lock
      - Reduce time dequeue blocks root lock for enqueue
  - Still TXQ lock for every packet
    - Due to SKB's have assigned a TXQ on enqueue

Results: Bulk qdisc dequeue

- Bulk dequeue results
  - Cpu 1: 1,565,117 pps

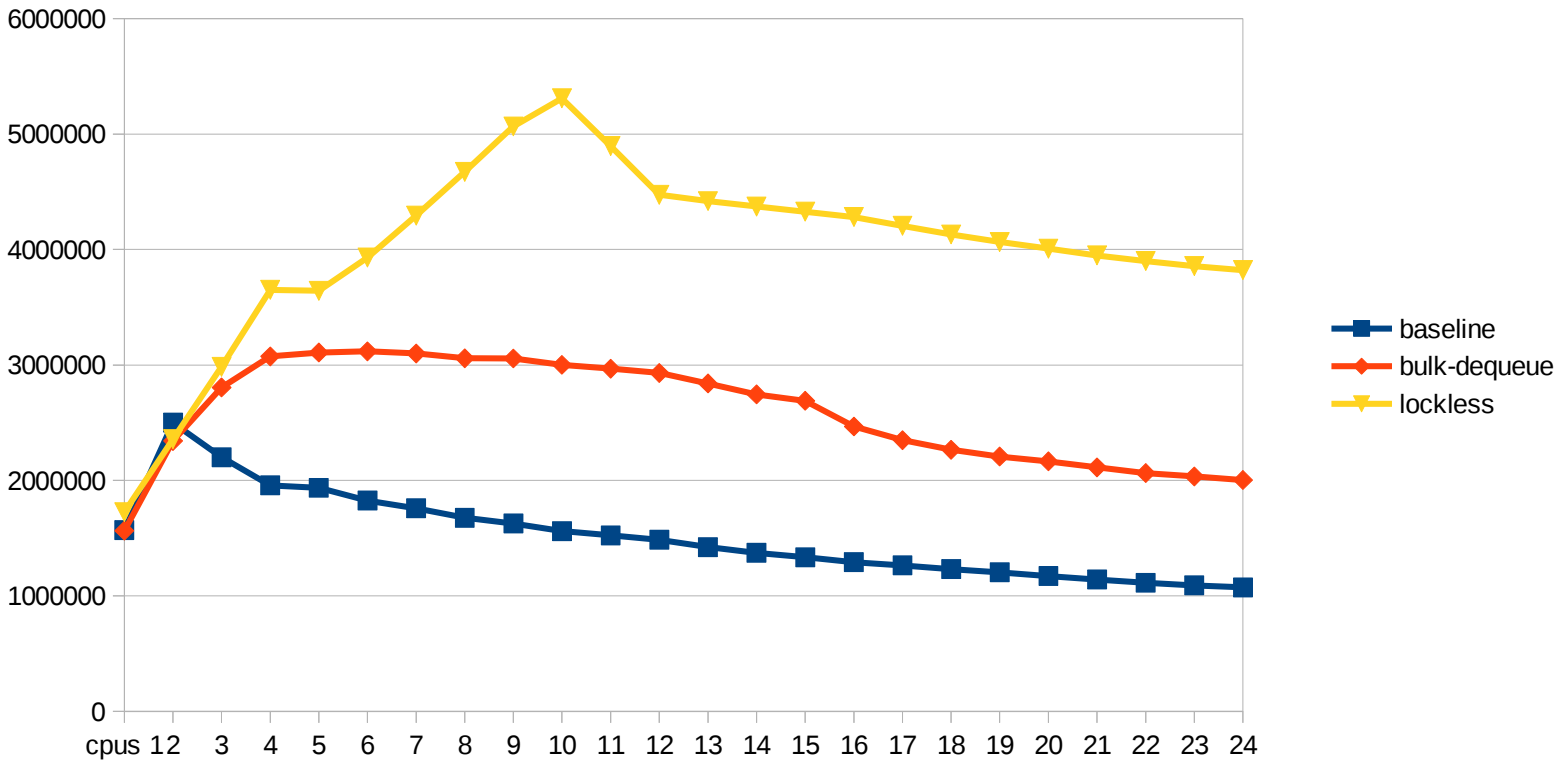


Implement: Lockless FIFO qdisc

- Needs John Fastabend qdisc RCU work
  - My hack just ignores stats
- Based on: Multi-Producer-Multi-Consumer ring queue
  - Requiring LOCK'ed cmpxchg
  - Bulk dequeue amortize cost of cmpxchg
  - Enqueue cmpxchg on every packet
    - Higher number of CPUs
      - Hotspot/sync-point on cmpxchg retry loop

Results: Lockless FIFO qdisc

- Lockless FIFO with bulk dequeue
  - Cpu 1: 1,726,467 pps



Comparing default use-case: Single CPU

- Lockless compared to qdisc NULL hack
  - 1566918 pps = 638.20 ns -- qdisc-path (clean kernel)
  - 1731000 pps = 577.70 ns -- qdisc NULL hack
  - 1726467 pps = 579.22 ns -- Lockless qdisc
  - Difference against NULL qdisc: *very close*
    - -4533 pps => +1.52 ns
  - Improvement against clean kernel:
    - +164082 pps => -58.98 ns
- Better than expected
  - Likely due to ignoring stats

Lockless: Forcing through MPMC

- Want to measure clean overhead of MPMC queue
  - Test: single CPU forced through MPMC
    - By removing TCQ_F_CAN_BYPASS flag
  - Expect added cost of two cmpxchg
- Results lockless qdisc w/o TCQ_F_CAN_BYPASS
  - 1726467 pps = 579.22 ns -- with BYPASS
  - 1703221 pps = 587.12 ns -- no BYPASS
  - -23246 pps => **+7.90 ns** -- Difference to bypass
- Were expecting to see 2x LOCK cost
  - But only see approx 1x LOCK cost

The End

- Open discussion
  - Where should we start?
    - Need John Fastabend's qdisc RCU work
  - Should we just bypass qdisc layer?

- We/Red Hat
  - Will allocate resources/persons to project
  - Open to collaborate

Extra

- Extra slides

Using TSQ

- TCP Small Queue (TSQ)
  - Use queue build up in TSQ
    - To send a bulk xmit
      - To take advantage of HW TXQ tail ptr update
    - Should we allow/use
      - Qdisc bulk enqueue
        - Detecting qdisc is empty allowing direct_xmit_bulk?