Session:

Linux packet processing performance improvements

TX bulking and qdisc layer

Jesper Dangaard Brouer (Red Hat)
John Fastabend (Intel)
John Ronciak (Intel)

Linux Plumbers Conference 16th Oct 2014

What will you learn?

- Unlocked full potential of driver (TX only)
- The xmit_more API for bulking
- Challenge bulking without adding latency
  - Qdisc layer bulk dequeue, depend on BQL
  - Existing aggregation GSO/GRO
- Qdisc locking is nasty
  - Amortization locking cost
  - Future: Lockless qdisc
- What about RX?

Unlocked: Driver TX potential

- Pktgen 14.8Mpps single core (10G wirespeed)
- Primary trick: *Bulking packet (descriptors) to HW*
- What is going on:
  - Defer tailptr write, which notifies HW
    - Very expensive write to none-cacheable mem
  - Hard to perf profile
    - Write to device
      - does not showup at MMIO point
      - Next LOCK op is likely “blamed”

API skb->xmit_more

- SKB extended with xmit_more indicator
  - Stack use this to indicate
  - another packet will be given immediately
    - After/when ->ndo_start_xmit() returns
- Driver usage
  - Unless TX queue filled
  - Simply add the packet to the TX queue
  - And defer the expensive indication to the HW

Challenge: Bulking without added latency

- Hard part:
  - **Use bulk API without adding latency**
- Principal: Only bulk when really needed
  - Based on solid indication from stack
- Do NOT speculative delay TX
  - Don't bet on packets arriving shortly
  - Hard to resist...
    - as benchmarking would look good

Use SKB lists for bulking

- Changed: Stack xmit layer
  - Adjusted to work with SKB lists
  - Simply use existing skb->next ptr
- E.g. See dev_hard_start_xmit()
  - skb->next ptr simply used as xmit_more indication
- Lock amortization
  - TXQ lock no-longer per packet cost
  - dev_hard_start_xmit() send entire SKB list
  - while holding TXQ lock (HARD_TX_LOCK)

Existing aggregation in stack GRO/GSO

- Stack already have packet aggregation facilities
  - GRO (Generic Receive Offload)
  - GSO (Generic Segmentation Offload)
  - TSO (TCP Segmentation Offload)
- Allowing bulking of these
  - Introduce no added latency
- Xmit layer adjustments allowed this
  - validate_xmit_skb() handles segmentation if needed

Qdisc layer bulk dequeue

- A queue in a qdisc
  - Very solid opportunity for bulking
    - Already delayed, easy to construct skb-list
- Rare case of reducing latency
  - Decreasing cost of dequeue (locks) and HW TX
    - Before: a per packet cost
    - Now: cost amortized over packets
- Qdisc locking have extra locking cost
  - Due to __QDISC___STATE_RUNNING state
  - Only single CPU run in dequeue (per qdisc)

Qdisc locking is nasty

- Always **6 LOCK** operations (6 * 8ns = 48ns)
  - **Lock** qdisc(root_lock) (also for direct xmit case)
    - Enqueue + possible Dequeue
      - Enqueue can exit if other CPU is running deq
      - Dequeue takes __QDISC___STATE_RUNNING
  - **Unlock** qdisc(root_lock)
  - **Lock** TXQ
    - Xmit to HW
  - **Unlock** TXQ
  - **Lock** qdisc(root_lock) (can release STATE_RUNNING)
    - Check for more/newly enqueued pkts
      - Softirq reschedule (if quota or need_sched)
  - **Unlock** qdisc(root_lock)

Qdisc bulking need BQL

- Only support qdisc bulking for BQL drivers
  - *Implement BQL in your driver now!*
- Needed to avoid overshooting NIC capacity
  - Overshooting cause requeue of packets
- Current qdisc layer requeue cause
  - Head-of-Line blocking
  - Future: better requeue in individual qdiscs?
- Extensive experiments show
  - BQL is very good at limiting requeues

Future work (qdisc)

- Qdisc proper requeue facility
  - Only implement for qdisc's that care
  - BQL might reduce requeues enough
- Allow bulk for qdisc one-to-many TXQ's
  - Current limited to flag TCQ_F_ONETXQUEUE
  - Requires some fixes to requeue system
- Test on small OpenWRT routers
  - CPU saving benefit might be larger

Future: Lockless qdisc

- Motivation for lockless qdisc (cmpxchg based)
  - Direct xmit case (qdisc len==0) “fast-path”
    - Still requires taking all 6 locks!
  - Enqueue cost reduced (qdisc len > 0)
    - from 16ns to 10ns
- Measurement show huge potential for saving
  - (lockless ring queue cmpxchg base implementation)
  - If TCQ_F_CAN_BYPASS saving 60ns
    - Difficult to implement 100% correct
  - Not allowing direct xmit case: saving 50ns

Qdisc RCU status

- Qdisc layer change
  - Needed to support lockless qdisc
  - All classifiers converted to RCU
  - Bstats/qstats per CPU
    - Do we want xmit stats per cpu?

Ingress qdisc

- Audit RCU paths one more time.
- Remove ingress qdisc lock

What about RX?

- TX looks good now
  - How do we fix RX?
- Experiments show
  - Highly tuned setup RX max 6.5Mpps
  - Forward test, single CPU only 1-2Mpps
- Alexie started optimizing the RX path
  - from 6.5 Mpps to 9.4 Mpps
    - via build_skb() and skb prefetch tunning

The End

- Thanks
  - Getting to this level of performance have been the jointed work and feedback from many people
- Download slides here:
  - http://people.netfilter.org/hawk/presentations/
- Discussion...