



Next steps for Linux Network stack approaching 100Gbit/s

Jesper Dangaard Brouer
Principal Engineer, Red Hat

The Camp, Juli 26, 2016
License: CC-BY-SA

Introduction

- Next steps for Linux Network stack
 - Approaching 100Gbit/s HW speeds
 - Software stack is under pressure!
- Disclaimer: This is my bleeding edge “plan”
 - Most of this is not accepted upstream
 - And might never be...!
 - Challenging work ahead!
 - Encourage people:
 - Go solve these issue before me! ;-)



Overview: Topics

- MM-bulk – more use-cases
- RX path – multi-fold solutions needed
 - Drivers RX-ring prefetching
 - RX bundles towards netstack
 - Page-pool
 - Make RX pages writable
 - Revert DMA performance-tradeoff hacks
- TX xmit_more “powers” – not used in practice
- Qdisc – Redesign needed?
- XDP – eXpress Data Path



MM-bulk: Status

- Status: upstream since kernel 4.6
 - Bulk APIs for `kmem_cache` (SLAB+SLUB)
 - Netstack use bulk *free* of **SKBs** in NAPI-context
 - Generic `kfree_bulk` API
- Rejected: Netstack bulk *alloc* of SKBs
 - As number of RX packets were unknown



MM-bulk: More use-cases

- Network stack – more use-cases
 - Need explicit bulk free use from TCP stack
 - NAPI bulk free, not active for TCP (keep ref too long)
 - Use `kfree_bulk()` for `skb` → head
 - (when allocated with `kmalloc`)
 - Use bulk free API for qdisc delayed free
- RCU use-case
 - Use `kfree_bulk()` API for delayed RCU free
- Other kernel subsystems?



RX path: Missed driver opportunities

- NAPI already allow a level of RX bulking
 - Drivers (usually) get 64 packet budget (by napi_poll)
 - Drivers don't take advantage of bulk opportunity
- Missed RX opportunities:
 - Drivers process RX-ring 1-packet at the time
 - Call full network stack every time
 - Cause:
 - I-cache likely flushed, when returning to driver code
 - Stall on cache-miss reading packet (ethertype)
 - No knowledge about how many "ready" RX packets



RX path: Early driver pre-RX-loop

- If RX ring contains multiple "ready" packets
 - Means kernel was too slow (processing incoming packets)
 - Thus, switch into more efficient mode (bulking)
 - Dynamically scaling to load...
- Idea: Split driver RX-loop
 - Introduce a pre-RX-loop for counting and prefetching
- Purpose of driver pre-RX loop
 - Knowing number of packets: allow bulk alloc of SKBs
 - Prefetching to hide cache-miss



RX path: DDIO technology

- Intel Data Direct I/O Technology (DDIO)
 - HW essentially deliver packet data in L3-cache
 - Only avail on high-end E5-based servers
- Driver pre-RX loop
 - Prefetch part: simplified software version of DDIO
- Still benefit for DDIO CPUs
 - Bulk alloc of SKBs, saving
 - (Only) hide L3->L1 cache miss
 - Better I-cache usage in driver-code



RX path: RX bulking to netstack

- More controversial to deliver a "bundle" to netstack
 - (Driver pre-RX loop is contained inside driver)
 - Split of Driver and netstack code, optimize/split I-cache usage
- **RFC proposal** by Edward Cree
 - Drivers simply queue RX pkts on SKB list (no-prefetch RX loop)
 - Results very good:
 - First step, 10.2% improvement (simply loop in netstack)
 - Full approach, 25.6% improvement (list'ify upto ip_rcv)
 - Interesting, but upstream was not ready for this step
- More opportunities when netstack know bundle size
 - E.g. caching lookups, flush/free when bundle ends



RX-path: Issue RX page are read-only

- Most drivers have read-only RX pages
 - Cause more expensive SKB setup
 - 1) Alloc separate writable mem area
 - 2) Copy over RX packet headers
 - 3) Store `skb_shared_info` in writable-area
 - 4) Setup pointers and offsets, into RX page-"frag"
- Reason: Performance trade off
 - A) Page allocator is too slow
 - B) DMA-API expensive on some platforms (with IOMMU)
 - Hack: alloc and DMA map larger pages, and "chop-up" page
 - Side-effect: read-only RX page-frames
 - Due to unpredictable DMA unmap time



RX-path: Make RX pages writable

- Need to make RX pages writable
 - This implicit what Eric Dumazet means when saying:
"Drivers should use `build_skb()`"
- My solution is the page-pool
 - Address:
 - Page-allocator speed
 - As a specialized allocator require less checks
 - DMA IOMMU mapping cost
 - Keeping page mapped
 - Make writable
 - By predictable DMA unmap point



Page-pool: Design

- Idea presented at [MM-summit April 2016](#)
- Basic ideas for a page-pool
 - Pages are recycled back into originating pool
 - Creates a feedback loop, helps limit pages in pool
 - Drivers still need to handle `dma_sync` part
 - Page-pool handle `dma_map/unmap`
 - essentially: constructor and destructor calls
- Page free/return to page-pool, Either:
 - 1) SKB free knows and call page pool free, **or**
 - 2) `put_page()` handle via page flag



Page-pool: opportunity – feedback loop

- Today: Unbounded RX page allocations by drivers
 - Can cause OOM (Out-of-Memory) situations
 - Handled via `skb->truesize` and queue limits
- Page pool provides a **feedback loop**
 - (Given pages are recycled back to originating pool)
 - Allow bounding pages/memory allowed per RXq
 - Simple solution: configure fixed memory limit
 - Advanced solution, track steady-state
 - Can function as a “Circuit Breaker” (See [RFC draft link](#))



TX powers – background

- Solved TX bottleneck with `xmit_more` API
 - See: <http://netoptimizer.blogspot.dk/2014/10/unlocked-10gbps-tx-wirespeed-smallest.html>
- 10G wirespeed: Pktgen 14.8Mpps single core
 - Spinning same SKB (no mem allocs)
- Primary trick: Bulk packet (descriptors) to HW
 - Delays HW NIC tailptr write
- Activated via Qdisc bulk dequeue
 - Issue: hard to “activate”



TX powers – performance gain

- Only artificial benchmarks realize gain
 - like pktgen
- How big is the difference?
 - with pktgen, ixgbe, single core E5-2630 @2.30GHz
 - TX **2.9 Mpps** (clone_skb 0, burst 0) (343 nanosec)
 - ↑ Alloc+free SKB+page on for every packet
 - TX **6.6 Mpps** (clone_skb 10000) (151 nanosec)
 - ↑ x2 performance: Reuse same SKB 10000 times
 - TX **13.4 Mpps** (pktgen burst 32) (74 nanosec)
 - ↑ x2 performance: **Use xmit_more** with 32 packet bursts
 - Faster CPU can reach wirespeed 14.8 Mpps (single core)



TX powers – Issue

- Only realized for artificial benchmarks, like pktgen
- Issue: For practical use-cases
 - Very hard to "activate" qdisc bulk dequeue
 - Qdisc supporting bulk dequeue (were) limited
 - Eric Dumazet very recently extended to more Qdisc's
 - Need to hit HW bandwidth limit to “kick-in”
 - Seen TCP hit BW limit, result lower CPU utilization
 - Want to realized gain earlier.
 - Next-step: bulk enqueue



Qdisc: layer issues

- Issues with qdisc layer
 - Too many (6) lock operations
 - even for the empty queue case!
 - Bulk TX xmit_more "powers" hard to utilize
 - Bulk enqueue could mitigate situation
 - Enqueue and dequeue block each-other
 - Enqueue'ers starve the single dequeuer
 - "strange" heuristic for avoiding enqueue to starve dequeue
- Thanks: Other people are looking at this area
 - Eric Dumazet, Florian Westphal and John Fastabend



Qdisc: Time to redesign qdisc layer?

- Interesting solution in article:
 - "A Fast and Practical Software Packet Scheduling Architecture"
 - By: Luigi Rizzo <rizzo@iet.unipi.it>
- Main take-way: "arbiter" serialize enqueue+dequeue step
 - packets are "submitted" in parallel (lockless queues)
 - arbiter scans queues, and preform enqueue step
- Linux already have single dequeue process "scheme"
 - Could take role of arbiter
 - If submitter/enqueue see `qdisc_is_running()`
 - store packet in intermediate lockless queue
 - arbiter/dequeue will guarantee to pickup fast, call `enqueue()`



XDP: eXpress Data Path

- An **eXpress Data Path (XDP)** in kernel-space
 - The "packet-page" idea from NetDev1.1 "rebranded"
 - Thanks to: Tom Herbert, Alexei and Brenden Blanco, putting effort behind idea
- Performance is primary focus and concern
 - Need features: use normal stack delivery
- Very exciting: Allow comparison against DPDK
 - Same lower level handling as DPDK
 - Allow comparing "apples-to-apples"



XDP: What is it?

- Thin layer at lowest levels of SW network stack
 - Before allocating SKBs
 - Inside device drivers RX function
 - Operate directly on RX packet-pages
- XDP is NOT kernel bypass
 - Designed to work in concert with stack
- XDP - run-time programmability via "hook"
 - Current proposal: run eBPF program at hook point



XDP: Stages

- Project still young
 - First XDP-summit held June 23 (2016)
- Phases of the project:
 - 1) Fast DDoS filter **[achievable]**
 - 2) One-legged load-balance/forwarding
 - in-out-same-NIC **[doable]**
 - 3) More generic forwarding **[challenging]**
 - 4) RAW packet dump (steal packets) **[challenging]**
- XDP patchset V10 accepted Juli 20
 - Basic infrastructure for phase 1 and 2
 - Will appear in kernel 4.8



XDP: Performance evaluation

- Evaluated on Mellanox 40Gbit/s NICs (mlx4)
 - Single CPU (with DDIO) performance
 - 20 Mpps – Filter drop all (but read/touch data)
 - 12 Mpps – TX-bounce forward (TX bulking)
 - 10 Mpps – TX-bounce with udp+mac rewrite
 - Single CPU without DDIO (cache-misses)
 - TX-bounce with udp+mac rewrite:
 - 8.5Mpps – cache-miss
 - 12.3Mpps – RX prefetch loop trick
- Page allocator is now primary bottleneck
 - Page-pool should remove that bottleneck



Status: Linux perf improvements

- Linux performance, recent improvements
 - approx past 2 years:
- Lowest TX layer (single core, pktgen):
 - Started at: 4 Mpps → 14.8 Mpps (← max 10G wirespeed)
- Lowest RX layer (single core):
 - Started at: 6.4 Mpps → 12 Mpps (still experimental)
 - XDP: drop 20Mpps (looks like HW limit)
- IPv4-forwarding
 - Single core: 1 Mpps → 2 Mpps → (experiment) 2.5Mpps
 - Multi core : 6 Mpps → 12 Mpps (RHEL7.2 benchmark)
 - XDP single core TX-bounce fwd: 10Mpps



The end

- Exciting times for network performance!
 - Evaluation show XDP will be as fast as DPDK



EXTRA SLIDES



RPS – Bulk enqueue to remote CPU

- RPS = Recv Packet Steering
 - Software balancing of flows (to/across CPUs)
- Current RPS
 - Remote CPUs does bulk/list-splice “dequeue”
 - RX CPU does single packet “enqueue”
- Experiment (Prove-of-concept code)
 - 4 Mpps RX limit hit with RPS
 - 9Mpps doing bulk “enqueue” (flush when NAPI ends)
 - The “dequeue” CPU can still only handle 4 Mpps

