Part of:

Network Performance Workshop

Memory bottlenecks

Jesper Dangaard Brouer
Principal Engineer, Red Hat

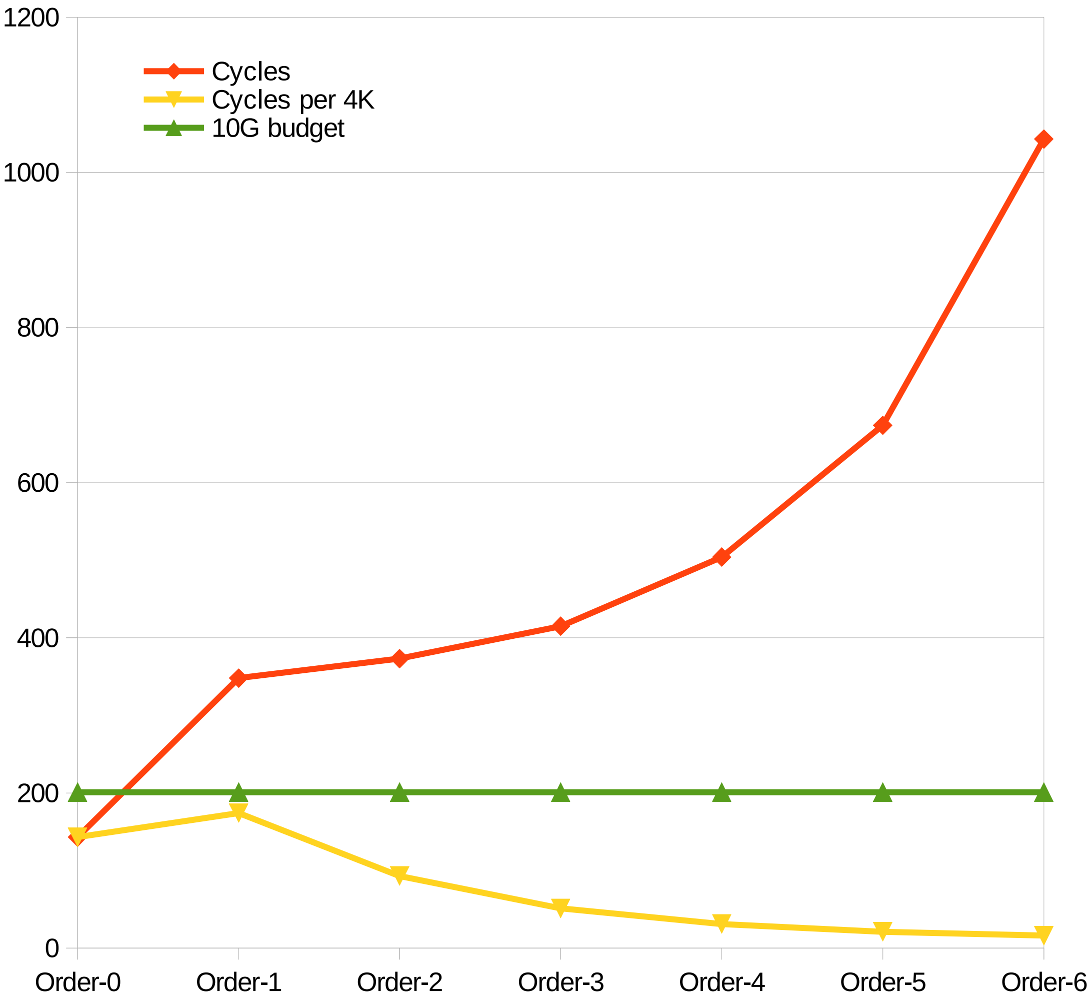
Date: April 2017
Venue: NetDevConf 2.1,
Montreal, Canada

Memory vs. Networking

- Network provoke bottlenecks in memory allocators
  - Lots of work needed in MM-area
- SLAB/SLUB area
  - Basically done via bulk APIs
- Page allocator current limiting XDP
  - Baseline performance too slow
  - Drivers implement page recycle caches
    - Can we generalize this?
    - And integrate this into page allocator?

Cost when page order increase (Kernel 4.11-rc1)

- Page allocator perf vs. size
  - Per CPU cache order-0
  - No cache > order-0
- Order to size:
  - 0=4K, 1=8K, 2=16K
- Yellow line
  - Amortize cost per 4K
  - Trick used by some drivers
  - Want to avoid this trick:
    - Attacker pin down memory
    - Bad for concurrent workload
    - Reclaim/compaction stalls



Issues with: Higher order pages

- Performance workaround:
  - Alloc larger order page, handout fragments
    - Amortize alloc cost over several packets
- Troublesome
  - 1. fast sometimes and other times require reclaim/compaction which can stall for prolonged periods of time.
  - 2. clever attacker can pin-down memory
    - Especially relevant for end-host TCP/IP use-case
  - 3. does not scale as well, concurrent workloads

Driver page recycling

- All high-speed NIC drivers do page recycling
  - Two reasons:
    - 1. page allocator is too slow
    - 2. Avoiding DMA mapping cost
- Different variations per driver
  - Want to generalize this
    - Every driver developer is reinventing a page recycle mechanism

Page pool: Generic recycle cache

- Basic concept for the page_pool
  - Pages are recycled back into originating pool
    - At put_page() time
  - Drivers still need to handle dma_sync part
  - Page-pool handle dma_map/unmap
    - essentially: constructor and destructor calls

The end

- kfree_bulk(7, slides);

Page pool: Generic solution, many advantages

- 5 features of a recycling page pool (per device):

1) Faster than page-allocator speed
   - As a specialized allocator require less checks
2) DMA IOMMU mapping cost removed
   - Keeping page mapped (credit to Alexei)
3) Make page writable
   - By predictable DMA unmap point
4) OOM protection at device level
   - Feedback-loop know #outstanding pages
5) Zero-copy RX, solving memory early demux
   - Depend on HW filters into RX queues