XDP – eXpress Data Path

Intro and future use-cases

Linux Kernel's fight against DPDK

Jesper Dangaard Brouer
Principal Engineer, Red Hat

Visiting One.com
Sep, 2016

Overview: Topics

- What is XDP – eXpress Data Path
- What is the proposed use-cases
- What can **you** imagine using this for?
  - No, this is not for every use-case!

Introduction

- 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
  - Target is competing with DPDK
  - No fancy features!
    - Need features: use normal stack delivery
- Disclaimer: This is my bleeding edge “plan”
  - Most of this is not accepted upstream
    - And might never be…!

XDP: What is XDP (eXpress Data Path)?

- 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"
  - Run eBPF program at hook point
  - Do you know what eBPF is?
    - User-defined, sandboxed bytecode executed by the kernel

XDP: data-plane responsibility “split”

- (Note: This is my personal abstract view of XDP)
- Split between kernel and eBPF
  - Kernel: fabric in charge of moving packets quickly
  - eBPF: logic decide action + read/write packet

XDP: Young project

- Project still young
  - First XDP-summit held June 23 (2016)
- XDP patchset V10 accepted Juli 20 (2016)
  - Basic infrastructure
    - Only implemented for one driver: mlx4
      - HW: ConnectX3-pro runs 10/40GbE
    - Will appear in kernel 4.8

XDP: Performance evaluation, crazy fast!!!

- 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
    - RX cache prefetch loop trick: 20 Mpps XDP_DROP

XDP: Packet based

- Packet based decision
  - (Currently) cannot store/propagate meta per packet
  - eBPF program can build arbitrary internal state (maps/hashes)
- Got **write** access to raw packet
  - Use-cases for modifying packets:
    - Add or pop encapsulation headers
    - Rewrite packet headers for forwarding/bouncing
    - Others?

XDP: Disclaimer

- Enabling XDP changes (RX ring) *memory model*
  - Needed to get write access to packet
  - Needed for fast drop (simple RX ring recycling)
  - Waste memory: Always alloc 4K (page) per RX packet
- Cause performance regression
  - When delivering packets to normal network stack
  - Due to bottleneck in page allocator
    - Working on page_pool project to remove this bottleneck
      - PoC code shows, faster than before!
  - Memory model waste can affect TCP throughput
    - Due to affecting skb->truesize

XDP: (FUTURE) per RX queue

- Current implementation
  - Same/single XDP program runs on ALL RX queues
- Plan: per RX queue attaching XDP programs
  - Use HW filters to direct traffic to RX queues
- Advantages:
  - More flexible, don’t “take” entire NIC
  - Can avoid changing memory model for all RX rings
    - Thus avoid performance regressions
  - Simpler XDP programs, with NIC HW filters
    - Less parsing of traffic as type is given by HW filter

XDP - actions

- Currently only implement 3 basic action
  1) XDP_PASS:
     - Pass into normal network stack (could be modified)
  2) XDP_DROP:
     - Very fast drop (recycle page in driver)
  3) XDP_TX:
     - Forward or TX-bounce back-out same interface
- I personally find "TX-bounce" very limiting
  - Cannot implement the DPDK router example

XDP - future actions

- XDP future actions:
  - XDP_FWD: Multi-port forwarding
    - Tricky settling on howto desc and return egress port
    - Depend on raw frame TX infrastructure in drivers
      - Getting lot of push-back upstream (strange!)
  - XDP capture to userspace (steal packet mode)
    - Faster tcpdump/RAW packets to userspace
      - Doable with a single copy
      - Zero-copy RX is tricky
        - Only possible with a combination of (1) dedicated RX HW rings, (2) HW filters, (3) separate page_pool recycling, and (4) premapping pages to userspace.

XDP port abstraction table proposal (FUTURE)

- Proposal for generalizing multi-port forwarding
  - How does eBPF “say” what egress “port” to use?
  - Bad approach: Tying a port to the netdev ifindex
    - Too Linux specific (Tom Herbert)
    - Limit the type of egress ports to be a netdev
    - XDP prog cannot be limited “allowed” set of ports
- XDP port abstraction table
  - Simply a “port” index lookup table
    - For “type” netdev: maps to ifindex (or net_device ptr)
    - For every “type” a new TX infrastructure needed

XDP use-cases

- Use-cases:
  - DDoS filtering
  - DDoS scrubbing box
  - Forwarding and load-balancing
  - Tunneling: encap/decap header handling
  - Sampling and monitoring tools
  - Faster packet dump (must steal packet)
  - Invent your own….?!
    - XDP infrastructure should support innovation

XDP: DDoS use-case

- First (obvious) use-case is DDoS filtering
  - Based on CloudFlares DNS/UDP filter (netdev 1.1)
- CloudFlare does kernel bypass
  - Single RX queue bypass into Netmap
  - Userspace (BPF) filter drop bad packets
  - Reinject good packets
- XDP can avoid reinject step
  - parse packet "inline" with eBPF

XDP: Types of DDoS

- DDoS filtering types:
  - Best suited for packet based filter decisions (L2 or L3)
  - eBPF could store historic state
    - Arbitrary advanced based on eBPF expressiveness
  - Use another tool for application layer attacks
- Really fast!
  - Realize: Can do wirespeed filtering of small packets
- Fast enough for?
  - Filtering DoS volume attacks on network edge?

XDP use-case: Load-balancing

- Facebook's use-case:
  - One-legged load-balancing
- Load-balancer without central LB-machine
  - Every machine (in cluster) is a load-balancer
    - If packet is not for localhost, XDP_TX forward to server responsible for terminating traffic.
- Same principle for: ILA-router
  - Based on IPv6 addr "split"
    - Identifier-Locator Addressing (ILA) for network virtualization
- Combine with Tunnel headers decap/encap

XDP use-case: Router

- Implement a router/forwarding data plane in eBPF
  - This is the DPDK prime example
- Depends on Multi-port TX (not implemented yet)
  - Need consistent design of
    - How to represent egress devices/ports?

XDP use-case: L2 learning bridge

- Assuming
  - Multi-port TX have been implemented
  - With port design accessible across XDP programs
- Natural step: L2 learning bridge
  - Connect/attach to bridge
    - Register (ingress) port + Load eBPF program
  - Flexibility of port design
    - Determine types of ports that can be attached
  - Ingress traffic builds FIB (Forward Information Base)
    - FIB lookup table is eBPF shared with a bpf-map.
    - Need kernel-side extension: Flood/broadcast on all ports

What are your XDP use-cases?

- Discuss what XDP could be used for?

XDP use-case: Bridge + Virtual machines

- Use-case: delivery into virtual machines (VM)
  - Depend on extending e.g. vhost-net with XDP compatible xmit function
- Combine L2-bridge with VM ports
  - L2-bridge is a known technology
  - VMs have a way of communicating
  - and discovery of each-other
- (eBPF could do arbitrary matching of VM)
  - save that idea for another time...

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

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 recycles 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)

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