XDP Infrastructure Development

"Internal" NetConf presentation

Jesper Dangaard Brouer
Principal Engineer, Red Hat

Date: April 2017
Venue: NetConf, Toronto (Canada)

Introduction

- This presentation is only relevant to
  - Infrastructure developers of XDP

- Presentation is about missing XDP features
  - As a ground for discussion

Since NetConf in Tokyo (Oct 2016)

- Success for: XDP_DROP and XDP_TX
- XDP being deployed in production
  - XDP_DROP big within DDoS use-case
    - Facebook, Cloudflare, One.com
  - XDP_TX used for Load-Balancing at Facebook
- More drivers support XDP
  - Some support push/pop headers helper

XDP and eBPF as programmable policy

- XDP and eBPF really good combination
  - New era in user programmable networking
- Kernel side: responsible for moving packet fast
- eBPF side: maximum flexibility and opt-in
  - User programmable protocol and policies
  - User maintains own wacky policy
    - Kernel is free from maintaining this forever \o/
  - Only run program code you actually need
    - No need to run code everybody else once needed

Fundamental: RX batching is missing

- Most fundamental performance principal “bulking”
  - Everybody else use RX packet batching/bulking
  - Know bulk update TX-tailptr is essential (see xmit_more)
- XDP driver "local" actions:
  - XDP_DROP: small impact (as long as eBPF icache is hot)
  - XDP_TX: Already does "bulk" via tairptr/doorbell "flush"
- For XDP_REDIRECT bulking is also needed
  - I prefer a explicit bulk interface
    - but a flush or xmit_more interface is also an option
  - API discussion, because packet-pages "leaves" driver

Amazing micro-benchmarks

- Fake "same" action bench cause implicit bulking benefits
  - XDP micro-benchmarks looks amazing
  - Driver and eBPF code reuse Instruction-cache
    - (calling normal netstack will flush icache)
- Basic concept to make this true for intermixed traffic
  - NOT pass XDP_PASS up the netstack immediately
  - See XDP as stage *before* netstack gets called

RX stages - high level view

- I envision driver RX code categorize packets
  - (within same NAPI budget) split into:
    - XDP_DROP and XDP_TX (driver local actions)
    - XDP_REDIRECT (outside driver)
    - Netstack delivery

Currently avoiding memory layer

- Current XDP features **secret to performance**:
- XDP_{DROP,TX} avoid calling memory layer
  - Local driver page recycle tricks
- Upcoming multi-port “XDP_REDIRECT”
  - Cannot hide behind local driver recycling
  - Need more generic solution
    - Like page_pool proposal
    - Or opportunistic page recycling via refcnt’s

Are current XDP action dangerous?

- XDP can modify/mangle any part of packet
  - This is no different from packets coming from network
    - Network stack is robust to handle these
- XDP_DROP
  - Not that scary, XDP installers choice to drop packet
- XDP_TX
  - Install XDP on ingress, fairly safe allowing egress
  - Slightly scary network wise, can create loops
- XDP_REDIRECT - real scary
  - Moving into **bypass arena** outside driver
  - Need mechanism to restrict what can be bypass

XDP "ifindex" bypass concerns

- Concerned XDP_REDIRECT want to use ifindex'es
- WARNING: Using ifindex is bad because it
  - Allow bypass without any control
  - Once a XDP program is loaded
    - kernel cannot restrict interfaces (XDP program can "use")
- Simply solution: vport lookup table
  - XDP forward/redirect ports become opt-in
  - Adding ports can leave audit trail
  - About controlling XDP injection points (sandboxing)

Benefits of port lookup table infra

- Port table lookup infrastructure, gains:
  - allows kernel to restrict ports participating
  - allows extending to other types than netdevices/ifindex
  - allows broadcast to all ports in given table
  - multiple port tables easy VRF separation
- Clear split between kernel and eBPF/XDP-prog
  - eBPF get/have ingress port# and return egress port#
  - Kernel only have port table (can validate/restrict egress port)
    - Port have type on kernel side
  - eBPF use maps to build relation/policy between ports

Use-case: Servers with management net

- Use-case: Servers with separate NIC for management
  - HP-ilo, Dell-Drac and IPMI is separate physical NIC
  - NIC exported to OS, used for management traffic
  - Common practice to install firewall rules to limit NIC
- XDP can bypass any NIC limits (like firewall)
  - Sends directly to NIC ifindex
  - Attacker only need to find off-by-one error
    - In eBPF program logic to use false ifindex
  - Port table solves this

Use-case: Cloud services

- Use-case: Allow Guest-OS to send XDP prog to Host-OS
  - Cloud providers want separation between Guest-OSs
  - XDP program is provided by Guest-OS / customer
    - untrusted customer program
- Need mechanism to limit what ports XDP prog can use
  - Host-OS install XDP prog and
  - associate with port-table-id

Use-case: accelerate Linux bridging

- XDP use-case: Accelerate Linux bridging
  - Basic idea: allow XDP program to XDP_REDIRECT
    - based on lookup in bridge FIB mac-table
  - Bridge handle complicated cases via XDP_PASS
    - Arp, flood-discovery, timeout etc
  - Discuss: access to bridge table or maintain separate eBPF map?
- Also need port table
  - need ability to restrict exit/desc-ports
  - when ports are removed for bridge
    - else XDP program out-of-sync can violate bridge

Use-case: learning bridge in XDP

- A learning bridge in XDP
  - Fully implemented in XDP
  - Need flood (or broadcast) facility
    - knowledge of that ports it need to flood
  - Port table could help with this

Use-case: Debugging XDP programs

- Troubleshooting a XDP program is hard
  - Cannot tcpdump packets on outgoing device
  - At least allow port table to restrict egress ports
    - As a troubleshooting tool

Details: ingress port need as input?

- Discuss: (might be too detailed to discuss)
  - Should kernel provide (ingress) port# to XDP-prog?
  - Most natural BUT not 100% required
- Alternative: eBPF can be compiled with port# as constant
  - Annoying for distributed systems with central compile/deployment
- A kernel side broadcast facility
  - Need to know port# to avoid flooding out incoming port
    - Work-around: XDP/eBPF returns port-id to exclude(?)

XDP documentation

- Started to doc XDP project: https://prototype-kernel.readthedocs.io
  - Not the official doc:
    - Need to be accepted (and reviewed) into kernel tree
- Found doc for eBPF-coding is needed first
- See documentation as a collaboration tool
  - I'm not controlling project, just the secretary
  - Capture design spec
    - from summary of upstream discussions
- Basic requirements link

Discuss: 1-page per packet restriction

- The motivation comes from
  - by design XDP pages are not shared
  - avoid atomic refcnt for XDP_DROP and XDP_TX recycling
  - control cache-coherency state of struct-page
- Doing refcnt adds cost (which DPDK don't have)
    - Most optimal conditions: 1-refcnt cost 30 cycles x2 = 60 cycles
      - accept this overhead for supporting page sharing?
      - (and unblocking of Intel driver support)
    - Cross CPU atomic refcnt cost, approx 260 cycles
      - e.g. page shared between two packets, going to separate CPUs

End of slide show

- Did I miss something?
  - XDP version of AF_PACKET?
- Any other XDP related topics?

Extra slides

XDP: Generic hook

- Generic XDP hook postponed
  - Not getting accepted before
    - A use-case cannot be handled by eBPF
- Who want to use a generic hook?
  - Nf-tables?
  - Bridge code?

Use-case: accelerate Linux bridging

- **After** implementing "multi-port" XDP_REDIRECT
- XDP use-case: Accelerate Linux bridging
  - Basic idea: allow XDP program to XDP_REDIRECT based on lookup in bridge FIB mac-table
  - Unknown packets do "XDP_PASS"
    - bridge handle complicated cases (timeout etc) via XDP_PASS
  - Discuss: access to bridge table or maintain separate eBPF map?
- Also need vport table
  - need ability to restrict exit/desc-ports
  - when ports are removed for bridge
  - else XDP program out-of-sync can violate bridge

Secret to XDP performance: I-cache

- XDP *benchmarks* does not stress I-cache
  - Hiding behind:
    - Very small benchmark bpf programs
    - Bench does not show intermixed traffic
- Once XDP programs get bigger
  - Running into I-cache misses
    - eBPF progs with tail calls
- Solution: Work on packet-array vector
  - Don’t intermix traffic XDP/netstack traffic

XDP userspace interface

- Bad semantics:
  - Blind load and override current XDP program
  - Leads to hard-to-debug issues for userspace
- XDP only query option is "bool"
  - Userspace don't know who's xdp_prog is running.
- Tools: like tc and iptables
  - Allow root to override/del
    - but have visibility to view current state
    - tc even have add/del/change/replace semantics

Improving XDP userspace interface?

- How can userspace detect eBPF prog changed?
- Programs can netlink monitor for "link" changes
  - Curr issue: replace will just show a xdp "true"
- Simple solution: static global ID counter
  - Attaching xdp_prog inc and return as id
  - netlink update contains ID
    - Give ability to identify/debug issues
- Could add/del/change/replace semantics be useful?
  - Acronym CRUD (Create, Read, Update and Delete)

XDP features and versions?

- How to handle: Capabilities negotiation?
  - Both driver and userspace tool need to
    - have concept of features/capabilities
- How do we handle adding new features?
  - and new versions of features?
- Current upstream solution assume:
  - that XDP_DROP and XDP_TX is always supported
  - XDP_DROP could be useful separately
    - and significantly easier to implement (e.g. e1000)
  - XDP_TX difficult on HW with a single TXq
    - Mess with netstack interaction e.g. BQL and fairness

Other API issues

- VLAN issue, only discussed, never implemented
  - AFAIKR VLAN ids should always be inlined in packet
    - Implies disabling HW features, when loading XDP
      - (Hint: not implemented…)

XDP prog per RX-queue

- Why a XDP program per RX-queue
  - Flexibility, do not monopolize entire NIC
- Performance issue:
  - XDP change memory model of RX queue
    - packet per page (trading memory for speed).
  - Cause perf regressions, for normal stack delivery
    - (1) bottleneck in page allocator (fixable via page_pool)
    - (2) skb→truesize increase, affecting TCP window
  - Thus, limit XDP to some RX queues, allow
    - Only affect traffic that really needs XDP

XDP: HW provided protocol offsets

- Most HW provide protocol offset in descriptor
  - E.g. Willy Tarreau "NDIV" solution use it
  - Pass useful L2/L3/L4 offsets to application
    - save it from parsing packets
    - (Presented in 2014 link)

- Would it be useful for XDP?
  - Find most efficient way to pass info to eBPF

XDP: Network Device Operation for "raw" xmit

- For multi-port TX
  - net_device extend with NDO for "raw" page transmit
    - Please: Bulking from day-0
      - Even if solving: lockless access to remote queue
  - Exposing TX queue number or not?
    - Likely best to hide TXq behind API
  - vhost_net
    - Can we “raw” transmit to a guest OS?
      - V1: Copy packet
      - V2: RX Zero-copy via dedicated RXq + page_pool