XDP - eXpress Data Path

XDP: a new fast and programmable network layer

Jesper Dangaard Brouer

Principal Engineer, Red Hat Inc.

Kernel Recipes

Paris, France, Sep 2018

What will **you** learn?

What will you actually get out of listening to this talk?

Learn that:

- Linux got a new fast and programmable network layer
  - And touch upon the basic building blocks
- How this XDP technology got motivated upstream
- Why eBPF is such a perfect match for XDP
- Demystify: Why this XDP technology is so fast!
- How XDP achieves "hidden" RX bulking
- Why XDP-redirect is a novel approach

Anti slide: What you will NOT learn

We cannot cover everything today...

XDP is about providing eBPF programmable superpowers to end-users

- But in this talk you will not learn how to code eBPF

Motivated and dedicated to: Making eBPF programming more easily consumable

- But NOT in this talk...
- ... go watch many of my other talks! :-)
  - Like the tutorial: XDP for the Rest of Us

Single slide intro: What is XDP?

Compressed slide about XDP ... cover the details later

XDP basically: New layer in the kernel network stack

- Before allocating the SKB
  - Driver level hook at DMA level
- Means: Competing at the same "layer" as DPDK / netmap
- Super fast, due to
  - Take action/decision earlier (e.g. skip some network layers)
  - No-memory allocations
- Not kernel bypass, data-plane is kept inside kernel
  - via BPF: makes early network stack run-time programmable
  - Cooperates with kernel

Next slides

What! - New layer in kernel networking?!

How did you motivate this upstream!?!

Motivation through competition

Competition was helpful to force us to find an upstream kernel alternative

Kernel bypass solutions: Like DPDK, netmap, PF_RING

- Show they are 10x faster than Linux kernel netstack
- This is true: FOR THEIR SPECIFIC USE-CASES

This lead to WRONG conclusions and quotes:

- Wrong: "Kernel is inherently too slow to handle networking"

Got motivated by showing this is WRONG

- Acknowledge competition, XDP was late to the game (2016 initial idea)
  - Netmap 2011 (SIGCOMM), DPDK 2013 (goes open-source?), PF_RING 2010 (earliest ref found)

Unfair comparison

Comparing apples and bananas

It is all about the use-case

- Kernel bypass benchmarks show L2/L3 use-cases
- Linux netstack assumes socket delivery use-case

Linux network stack was build with socket-delivery use-case in focus

- Naming of container for packet data structure says is all:
  - SKB → sk_buff → (originates from) socket buffer
  - Thus, netstack takes this cost upfront (at driver level alloc SKB)

XDP design goals

Operate at same L2/L3 level, allows for apple to apple comparison

XDP basically change the picture: Operate at same level

- Introduce layer before allocating the SKB
- Hook at driver level (running eBPF)

XDP goals

- Close the performance gap to kernel-bypass solutions
  - Not a goal to be faster
- Provide in-kernel alternative, that is more flexible
  - Don't steal the entire NIC
- Work in concert with existing network stack

Next slides

Show me some benchmarks!

Benchmark: Drop packet

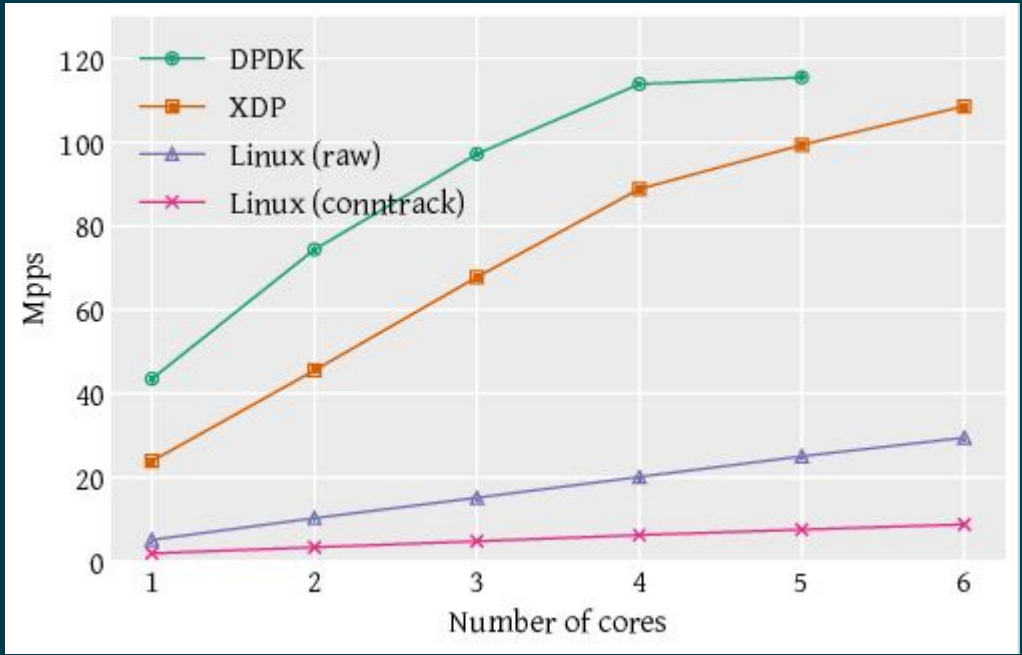
Driver: mlx5,
HW: 100Gbit/s ConnectX-5 Ex.
CPU: E5-1650v4

Comparing dropping packet:
DPDK vs. XDP vs Linux

- XDP follow DPDK line,

  offset is due to driver indirect calls, calculated to 16 nanosec offset

- Linux scales perfectly, conntrack overhead is significant
- HW/driver does PCIe compression to avoid PCIe limitations

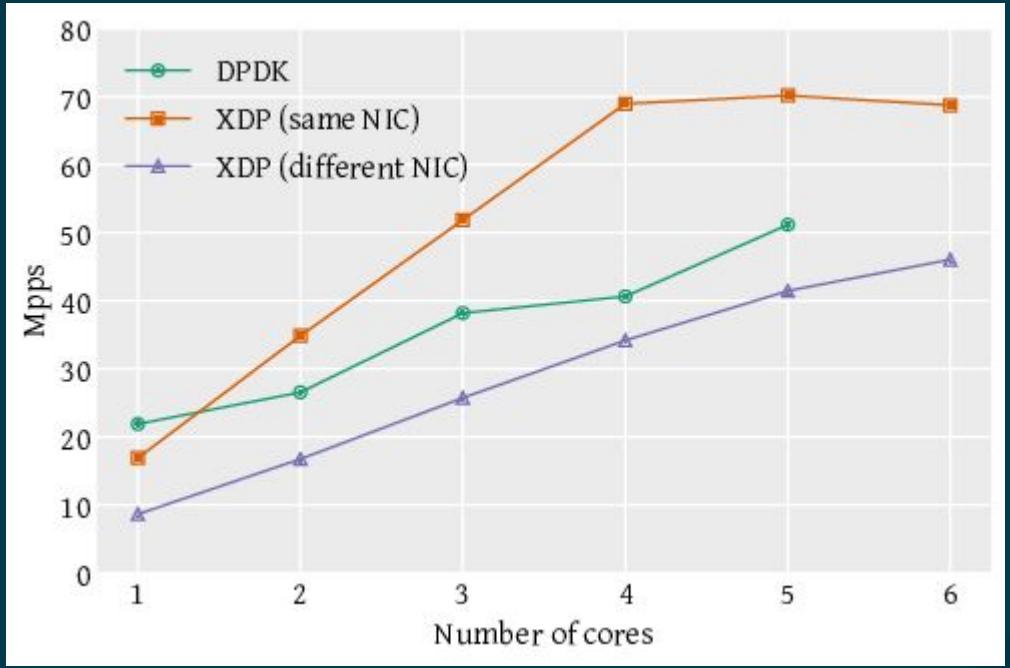


Benchmark: L2 forwarding

Driver: mlx5,
HW: 100Gbit/s ConnectX-5 Ex.
CPU: E5-1650v4

Comparing L2 forward packets

- XDP-same-NIC uses XDP_TX, which bounce packet inside NIC driver
- XDP-different-NIC use XDP_REDIRECT, which is more advanced (further optimization possible)
- Shows XDP can be faster than DPDK, in certain cases/modes



Next slides

Sounds cool!

Tell me more about: Basic design and building blocks

XDP actions and cooperation

What are the basic building blocks I can use?

BPF program return an action or verdict

- XDP_DROP, XDP_PASS, XDP_TX, XDP_ABORTED, XDP_REDIRECT

How to cooperate with network stack

- Pop/push or modify headers: Change RX-handler kernel use
  - e.g. handle protocol unknown to running kernel
- Can propagate 32Bytes meta-data from XDP stage to network stack
  - TC (clsbpf) hook can use meta-data, e.g. set SKB mark

Design: XDP: data-plane and control-plane

Overall design

Data-plane: inside kernel, split into:

- Kernel-core: Fabric in charge of moving packets quickly
- In-kernel BPF program:
  - Policy logic decide action
  - Read/write access to packet

Control-plane: Userspace

- Userspace load BPF program
- Can control program via changing BPF maps
- Everything goes through bpf system call

Why kernel developers should love BPF

How BPF avoids creating a new kernel ABI for every new user-invented policy decision

BPF is sandboxed code running inside kernel (XDP only loaded by root)

- A given kernel BPF hook just define:
  - possible actions and limit helpers (that can lookup or change kernel state)

Users get programmable policies (within these limits)

- Userspace "control-plane" API tied to userspace app (not kernel API)
  - likely via modifying a BPF-map
- No longer need a kernel ABI
  - like sysctl/procfs/ioctls etc.

Next slides

Why XDP_REDIRECT is so interesting?!

XDP action XDP_REDIRECT

First lets cover the basics, net_device redirect (... later, more flexibility in maps)
Avail since kernel v4.14, but drivers slower to adapt

XDP got action code XDP_REDIRECT (that drivers must implement)

- In basic form: Redirecting RAW frames out another net_device/ifindex
- Egress driver: implement `ndo_xdp_xmit` (and `ndo_xdp_flush`)

Performance low without using a map for redirect (single CPU core numbers, ixgbe):

- Using helper: bpf_redirect = 7.5 Mpps
- Using helper: bpf_redirect_map = 13.0 Mpps

What is going on?

- Using redirect maps is a HUGE performance boost, why!?

Novel: redirect using BPF maps

Why is it so brilliant to use BPF maps for redirecting?

Basic design: Simplify changes needed in drivers

- Name “redirect” is more generic, than “forwarding”

First trick: Hide RX bulking from driver code via MAP

- Driver still processes packets one at a time - calling `xdp_do_redirect`
- End of driver NAPI poll routine “flush” (max 64 packets) - call `xdp_do_flush_map`
- Thus, bulking via e.g. delaying expensive NIC tailptr/doorbell

Second trick: invent new types of redirects easy

- Without changing any driver code!
- Hopefully last XDP action code(?)

Gist: Driver XDP RX-processing (NAPI) loop

Demonstrate NAPI-poll call BPF prog per packet and end with MAP-flush

Basic code needed in Driver for supporting XDP:

```
1  while (desc_in_rx_ring && budget_left--) {
2   action = bpf_prog_run_xdp(xdp_prog, xdp_buff);
3   /* helper bpf_redirect_map have set map (and index)via this_cpu_ptr  */
4   switch (action) {
5    case XDP_PASS:      break;
6    case XDP_TX:        res = driver_local_xmit_xdp_ring(adapter, xdp_buff); break;
7    case XDP_REDIRECT:  res = xdp_do_redirect(netdev, xdp_buff, xdp_prog); break;
8    default:            bpf_warn_invalid_xdp_action(action); /* fallthrough */
9    case XDP_ABORTED:   trace_xdp_exception(netdev, xdp_prog, action); /* fallthrough */
10   case XDP_DROP:      res = DRV_XDP_CONSUMED; break;
11  } /* left out acting on res */
12 } /* End of NAPI-poll */
13 xdp_do_flush_map(); /* Bulk chosen by map, can store xdf_frame for flushing */
14 driver_local_XDP_TX_flush();
```

Redirect map types

What kind of redirects are people inventing?!

The “devmap”: BPF_MAP_TYPE_DEVMAP

- Contains net_devices, userspace adds them via ifindex to map-index

The “cpumap”: BPF_MAP_TYPE_CPUMAP

- Allow redirecting RAW xdp frames to remote CPU
  - SKB is created on remote CPU, and normal network stack invoked
- The map-index is the CPU number (the value is queue size)

AF_XDP - “xskmap”: BPF_MAP_TYPE_XSKMAP

- Allow redirecting RAW xdp frames into userspace
  - via new Address Family socket type: AF_XDP
  - Very close to ‘netmap’ design, keeping drivers in kernel-space
  - Ask Björn Töpel who implemented it… in audience

Next slides

Spectre V2 killed XDP performance

Performance issue: Spectre (variant 2)

CONFIG_RETPOLINE and newer GCC compiler
- for stopping Spectre (variant 2) CPU side-channel attacks

Hey, you killed my XDP performance! (Retpoline tricks for indirect calls)

- Still processing 6 Mpps per CPU core
- But could do approx 13 Mpps before!

Initial through it was `net_device->ndo_xdp_xmit` call

- Implemented redirect bulking, but only helped a little

Real pitfall: DMA API use indirect function call pointers

- Christoph Hellwig PoC patch show perf return to approx 10 Mpps

Thus, solutions in the pipeline...

Next slides

Questions asked by NDIV (HAproxy) in:

“Challenges migrating from NDIV to XDP”

NetDev Conf 0x12 (slides and video)

NDIV need callback rx_done()

NDIV have callback rx_done() per packet batch, at end of RX loop

NDIV callback `rx_done()` per packet batch, at end of RX loop

- XDP do have this, but part of BPF map abstraction
- For XDP, correspond to: `xdp_do_flush_map`

In theory, NDIV could be implemented with BPF as

- Move match/filter part into BPF prog, that return action XDP_REDIRECT
- New redirect BPF map type for NDIV
  - (Ugly) could do mem alloc for NDIV “out-packet(s)”
  - (Ugly) could invoke `ndiv->handle_rx(xdp_frame)`
- This map type, can receive event RX-done via xdp_do_flush_map

XDP hook at TX?

Could not find the XDP hook at TX
NDIV have `ndiv->handle_tx(ndiv, skb)`

There is no TX hook in XDP, why?

- For packets arriving from netstack
  - makes no-sense performance-wise (too late, SKB already alloc)
- BPF hook for TX is available
  - Use TC (clsbpf) egress BPF-prog hook (takes SKB like your handler)

(Disclaimer only idea:) Maybe do XDP bpf_prog invoke if ndo_xdp_xmit() fails

- Use-case: react to XDP-redirect TX overflow
  - E.g. allow new XDP action, like another redirect target (load-balance)

End slide

... Questions?


plus.google.com/+JesperDangaardBrouer


facebook.com/brouer


linkedin.com/in/brouer


twitter.com/JesperBrouer


youtube.com/channel/UCSyplUCgtI42z63soRMONng

Thanks to all contributors

XDP + BPF combined effort of many people

- Alexei Starovoitov
- Daniel Borkmann
- Brenden Blanco
- Tom Herbert
- John Fastabend
- Martin KaFai Lau
- Jakub Kicinski
- Jason Wang
- Andy Gospodarek
- Thomas Graf
- Michael Chan (bnxt_en)
- Saeed Mahameed (mlx5)
- Tariq Toukan (mlx4)
- Björn Töpel (i40e + AF_XDP)
- Magnus Karlsson (AF_XDP)
- Yuval Mintz (qede)
- Sunil Goutham (thunderx)
- Jason Wang (VM)
- Michael S. Tsirkin (ptr_ring)
- Edward Cree
- Toke Høiland-Jørgensen

Next slides

Extra slides if time permits...

Next slides

What is this CPUMAP redirect?

XDP_REDIRECT + cpumap

What is cpumap redirect?

Basic cpumap properties

- Enables redirection of XDP frames to remote CPUs
- Moved SKB allocation outside driver (could help simplify drivers)

Scalability and isolation mechanism

- Allows isolating/decouple driver XDP layer from network stack
  - Don't delay XDP by deep call into network stack
- Enables DDoS protection on end-hosts (that run services)
  - XDP fast-enough to avoid packet drops happen in HW NICs

Another use-case: Fix NIC-HW RSS/RX-hash broken/uneven CPU distribution

- Proto unknown to HW: e.g. VXLAN and double-tagged VLANs
- Suricata use this feature

Cpumap redirect: CPU scaling

Tricky part getting cross CPU delivery fast-enough

Cpumap architecture: Every slot in array-map: dest-CPU

- MPSC (Multi Producer Single Consumer) model: per dest-CPU
  - Multiple RX-queue CPUs can enqueue to single dest-CPU
- Fast per CPU enqueue store (for now) 8 packets
  - Amortized enqueue cost to shared ptr_ring queue via bulk-enq
- Lockless dequeue, via pinning kthread CPU and disallow ptr_ring resize

Important properties from main shared queue ptr_ring (cyclic array based)

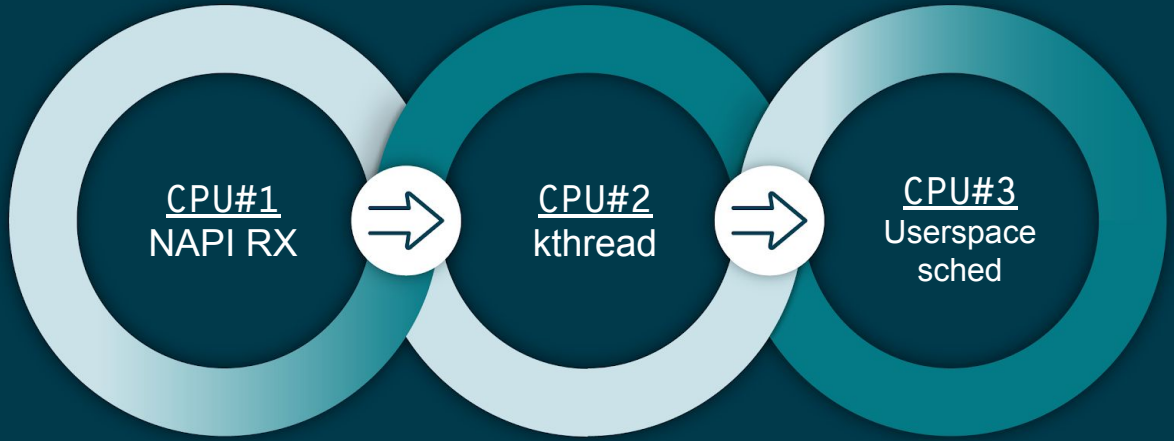
- Enqueue+dequeue don't share cache-line for synchronization
  - Synchronization happens based on elements
  - In queue almost full case, avoid cache-line bouncing
  - In queue almost empty case, reduce cache-line bouncing via bulk-enq

CPU scheduling via cpumap

Queuing and scheduling in cpumap

Hint: Same CPU sched possible

- But adjust /proc/sys/kernel/sched_wakeup_granularity_ns



Next slides

Recent changes to XDP core

Recent change: Information per RX-queue

Recent change in: kernel v4.16

Long standing request: separate BPF programs per RX queue

- This is not likely to happen… because

Solution instead: provide info per RX queue (`xdp_rxq_info`)

- Info: ingress net_device (Exposed as: `ctx->ingress_ifindex`)
- Info: ingress RX-queue number (Exposed as: `ctx->rx_queue_index`)

Thus, NIC level XDP/bpf program can instead filter on rx_queue_index

Recent change: queuing via xdp_frame

Very recent changes: only accepted in net-next (to appear in v4.18)

XDP_REDIRECT needs to queue XDP frames e.g. for bulking

- Queuing open-coded for both cpumap and tun-driver
- Generalize/standardize into struct xdp_frame
- Store info in top of XDP frame headroom (reserved)
  - Avoids allocating memory

Recent change: Memory return API

Very recent changes: only accepted in net-next (to appear in v4.18)

API for how redirected frames are freed or "returned"

- XDP frames are returned to originating RX driver
- Furthermore: this happens per RX-queue level (extended `xdp_rxq_info`)

This allows driver to implement different memory models per RX-queue

- E.g. needed for AF_XDP zero-copy mode