XDP (eXpress Data Path) as a building block for other FOSS projects

Jesper Dangaard Brouer (Red Hat)
Magnus Karlsson (Intel)

FOSDEM 2019
Brussels, Feb 2019

Framing XDP

XDP: new in-kernel programmable (eBPF) layer before netstack

- Similar speeds as DPDK

XDP ensures that Linux networking stays relevant

- Operates at L2-L3, netstack is L4-L7

XDP is not first mover, but we believe XDP is different and better

- Killer feature: Integration with Linux kernel
- Flexible sharing of NIC resources

What is XDP?

XDP (eXpress Data Path) is a Linux in-kernel fast-path

- New programmable layer in-front of traditional network stack
- Already accepted part of upstream kernels (and RHEL8)
- Operate at the same level and speeds as DPDK
- For L2-L3 use-cases: seeing x10 performance improvements!
- Can accelerate in-kernel L2-L3 use-cases (e.g. forwarding)

What is AF_XDP? (the Address Family XDP socket)

- Hybrid kernel-bypass facility, move selective frames out of kernel
- XDP/eBPF prog filters packets using REDIRECT into AF_XDP socket
- Delivers raw L2 frames into userspace

Why is XDP needed?

This is about the Kernel networking stack staying relevant

- For emerging use-cases and areas

Linux networking stack optimized for layers L4-L7

- Missing something to address L2-L3 use-cases

XDP operate at layers L2-L3

If you forgot OSI model:

- L2=Ethernet
- L3=IPv4/IPv6
- L4=TCP/UDP
- L7=Applications

Existing solutions: Not first mover

XDP is not first mover in this area

- But we believe XDP is different and better

Existing kernel bypass solutions:

- netmap (FreeBSD), DPDK (Intel/LF), PF_ring (ntop)
- maglev (Google), Onload (SolarFlare), Snabb

Commercial solutions similar to XDP:

- ndiv by HAproxy, product ALOHA

What makes XDP different and better?

Not bypass, but in-kernel fast-path

The killer feature of XDP is integration with Linux kernel,

- Leverages existing kernel infrastructure, eco-system and market position
- Programmable flexibility via eBPF sandboxing (kernel infra)
- Flexible sharing of NIC resources between Linux and XDP
- Cooperation with netstack via eBPF-helpers and fallback-handling
- No need to reinject packets (unlike bypass solutions)

AF_XDP for flexible kernel bypass

- Cooperate with use-cases needing fast raw frame access in userspace
- While leveraging existing kernel NIC drivers

XDP is a building block

Fundamental to understand that XDP is a building block

XDP is a building block

It is fundamental to understand

XDP is a component; a core facility provided by the kernel

- Put it together with other components to solve a task

eBPF (incl XDP) is not a product in itself

- Existing (and new) Open Source projects will use these eBPF components

Full potential comes when

- Combining XDP-eBPF with other eBPF-hooks and facilities
- To construct a “networking pipeline” via kernel components
- The Cilium project is a good example (container L4-L7 policy)

XDP use-cases

Areas and use-cases where XDP is already being used

Touch upon new potential and opportunities

- e.g. for Virtual Machines (VM) and Containers

Use-case: Anti-DDoS

The most obvious use case for XDP is anti-DDoS

Companies already deployed XDP in production for anti-DDoS

- Facebook, every packet goes through XDP for 1.5 years
- CloudFlare switched to XDP (changed NIC vendor due to XDP support!)

New potential: Protecting Containers and VMs

- Containers: Protect Kubernetes/OpenShift cluster with XDP
- VM: Host-OS protect Guest-OS'es via XDP
  - Work-in-progress: allow vhost/virtio_net; upload XDP to Host-OS

Use-case: L4 Load-balancer

Facebook was using the kernel Load-balancer IPVS

- Switched to using XDP instead: Reported x10 performance improvement
- Open Sourced their XDP load-balancer called katran

New potential: Host OS load-balancing to VMs and Containers

- VM: Phy-NIC can XDP_REDIRECT into Guest-NIC
  - driver tuntap queues XDP-raw frames to virtio_net; skip SKB in Host-OS
- Container: Phy-NIC can XDP_REDIRECT into veth (kernel v4.20)
  - driver veth allocs+builds SKB outside driver-code; speedup skip some code
  - veth can RE-redirect, allow building interesting proxy-solutions

Evolving XDP via leveraging existing solutions

XDP can (easily) be misused in the same way as kernel bypass solutions

Being smart about how XDP is integrated into existing Open Source solutions

- Leverage existing eco-systems e.g. for control plane setup

Evolving XDP via BPF-helpers

We should encourage adding helpers instead of duplicating data in BPF maps

Think of XDP as a software offload layer for the kernel netstack

- Simply setup and use the Linux netstack, but accelerate parts of it with XDP

IP routing good example: Access routing table from XDP via BPF helpers (v4.18)

- Let Linux handle routing (daemons) and neighbour lookups
- Talk at LPC-2018 (David Ahern): Leveraging Kernel Tables with XDP

Obvious next target: Bridge lookup helper

- Like IP routing: transparent XDP acceleration of bridge forwarding
  - Fallback for ARP lookups, flooding etc.
- Huge potential performance boost for Linux bridge use cases!

Transfer info between XDP and netstack

Ways to transfer info between XDP and netstack

- XDP can modify packet headers before netstack
  - Pop/push headers influence RX-handler in netstack
  - CloudFlare modifies MAC-src on sampled dropped packets
- XDP have 32 bytes metadata in front of payload
  - TC eBPF (cls_bpf) can read this, and update SKB fields
  - E.g. save XDP lookup and use in TC eBPF hook
  - AF_XDP raw frames have this metadata avail in front of payload

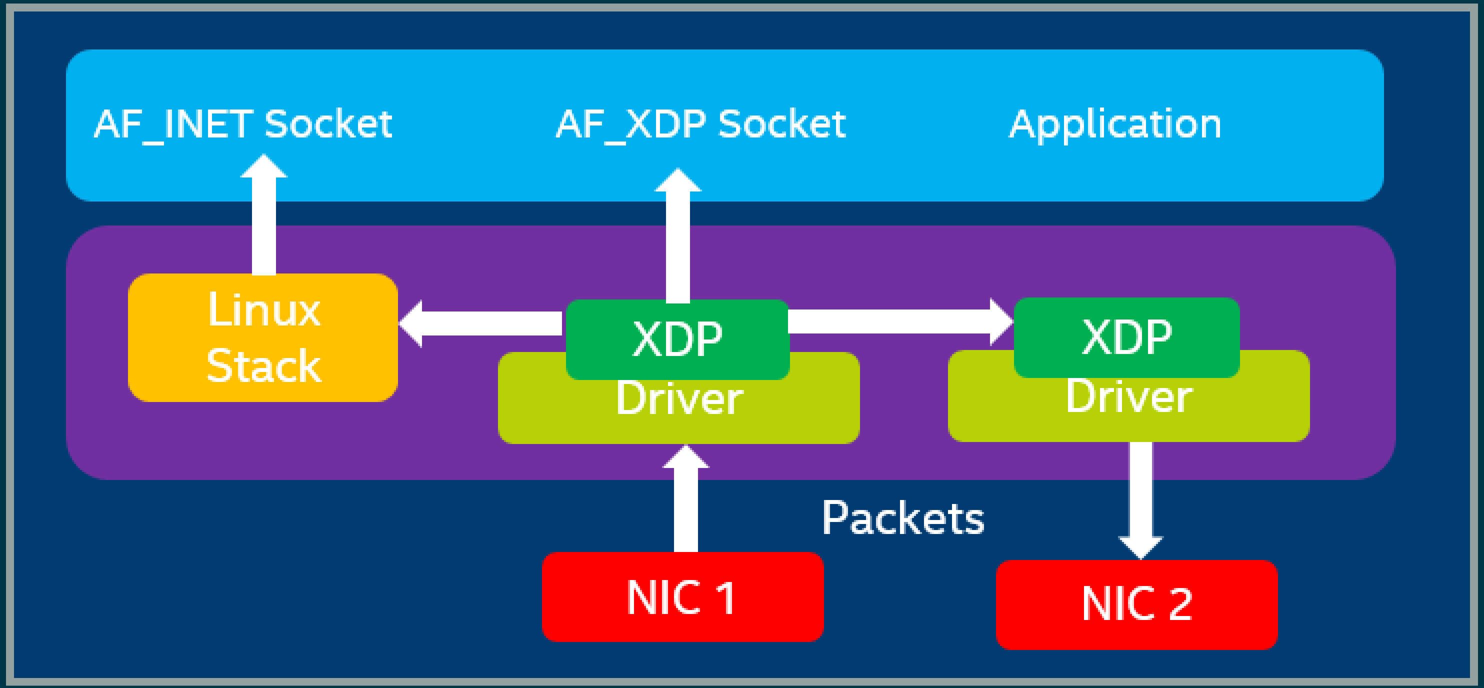
XDP integration with OVS

XDP/eBPF can integrate/offload Open vSwitch (OVS) in many ways

- VMware (William Tu) presented different options at LPC 2018:
  - Bringing the Power of eBPF to Open vSwitch
- TC eBPF, (re)implemented OVS in eBPF (performance limited)
- Offloading subset to XDP (issue: missing some BPF helpers)
- AF_XDP, huge performance gain

AF_XDP

AF_XDP Basics

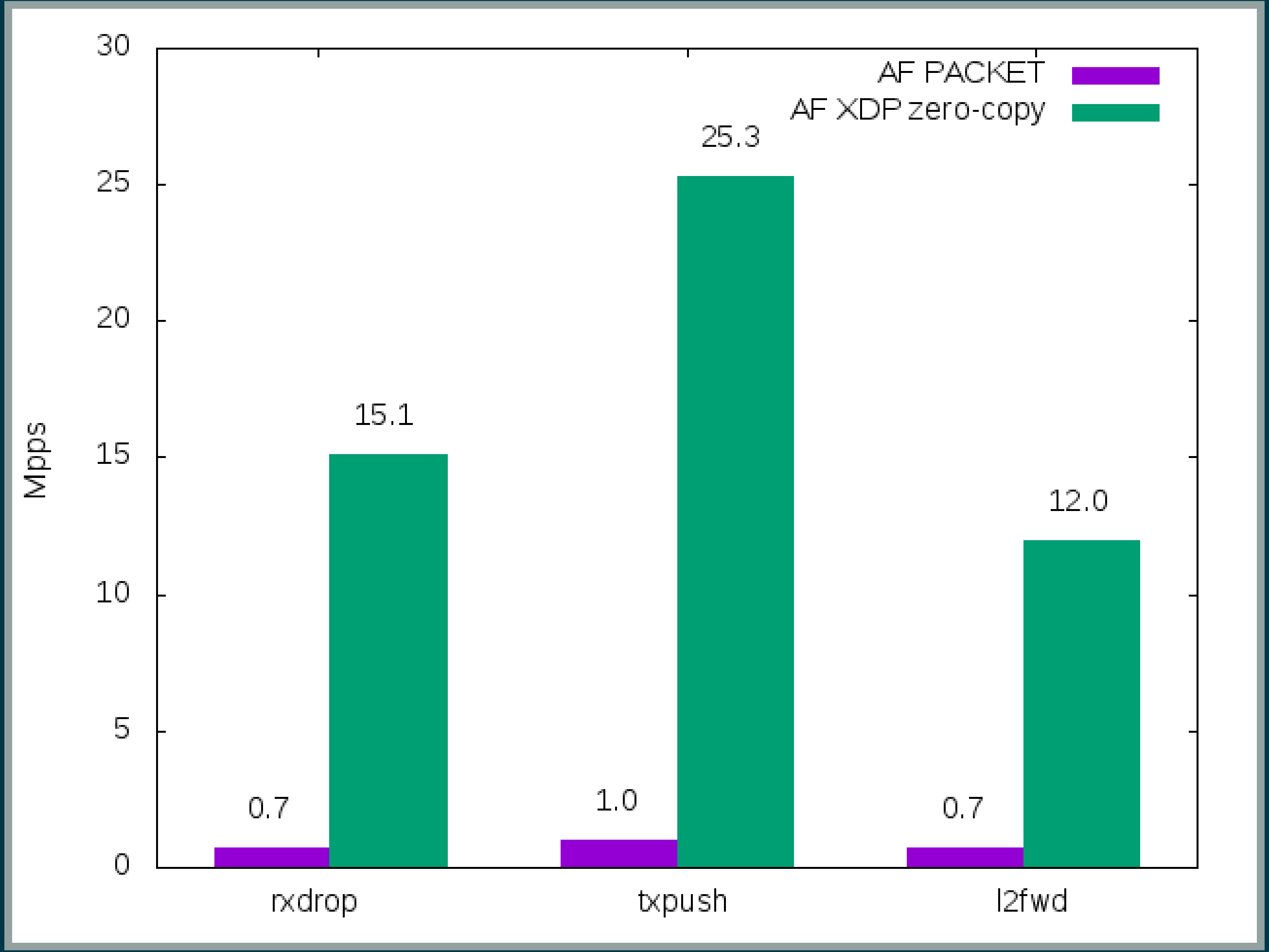


Performance

Experimental Methodology

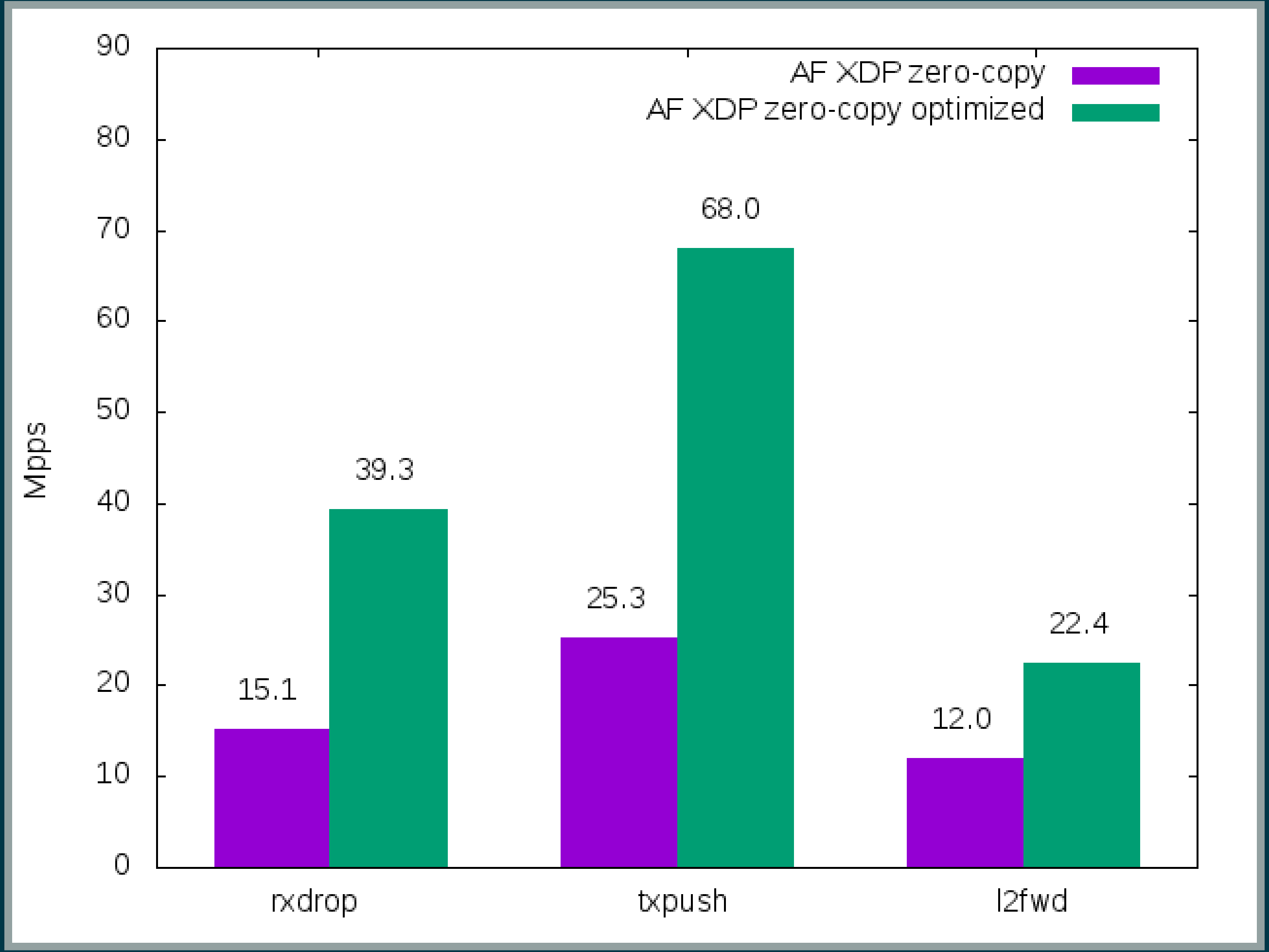
- Broadwell E5-2660 @ 2.7GHz (with DDIO = L3 payload delivery)
- Linux kernel 4.20
- Spectre and Meltdown mitigations on
- 2 i40e 40GBit/s NICs, 2 AF_XDP sockets
- Ixia load generator blasting at full 40 Gbit/s per NIC

Performance Linux 4.20



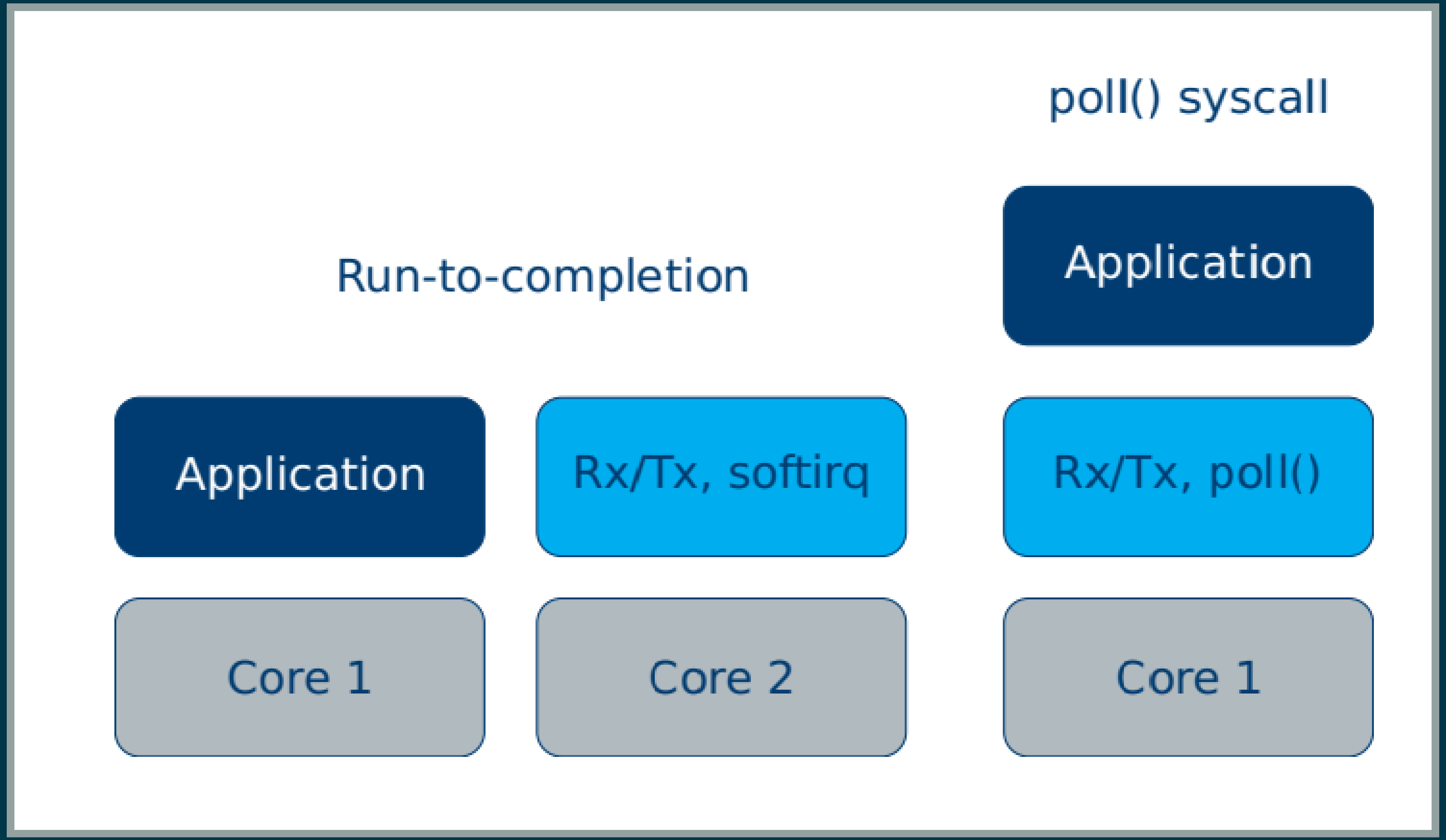
Huge improvement compared to AF_PACKET, more optimizations in pipeline

Performance with Optimization Patches

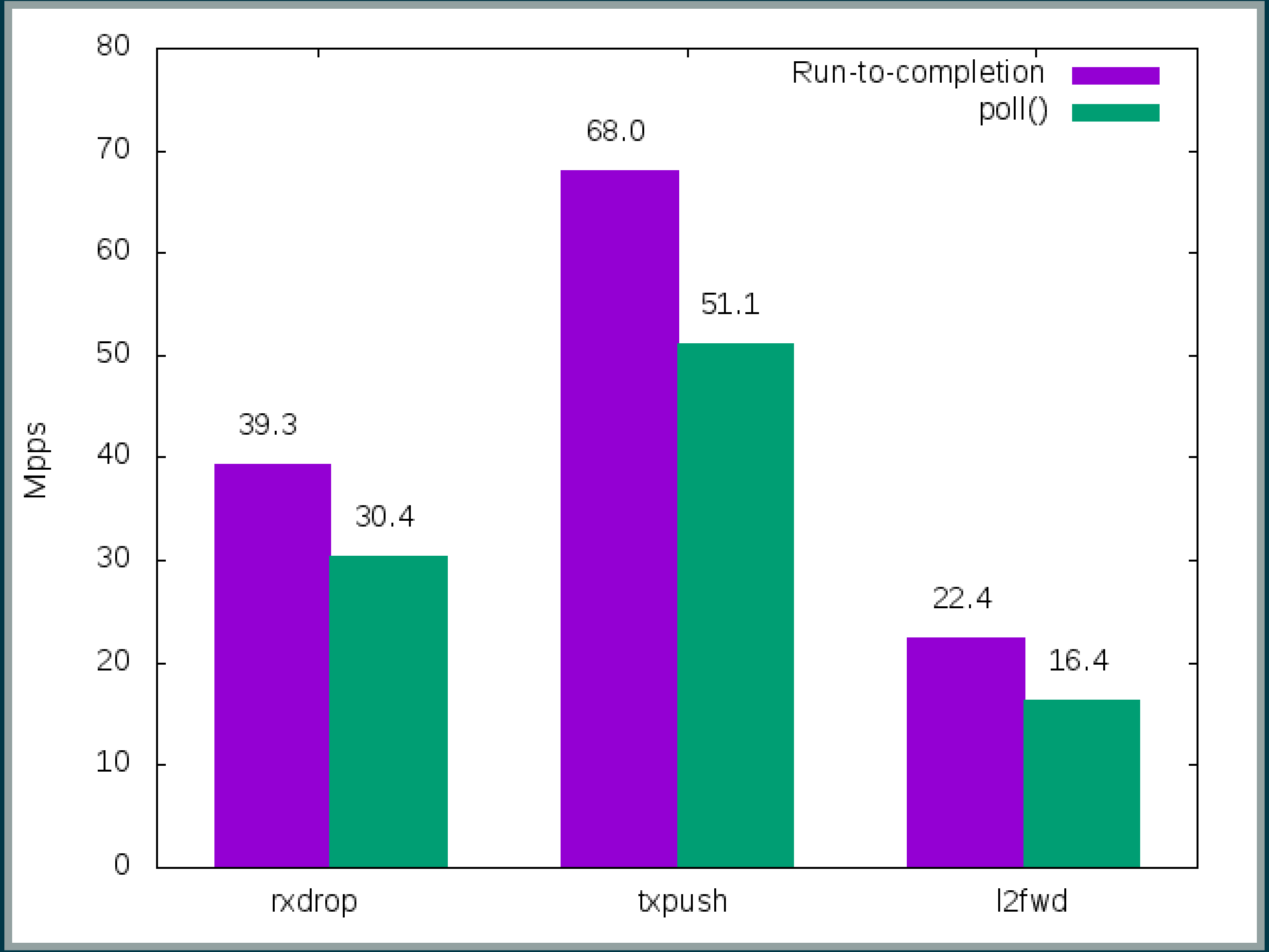


Details see LPC2018 talk: The Path to DPDK speeds for AF_XDP

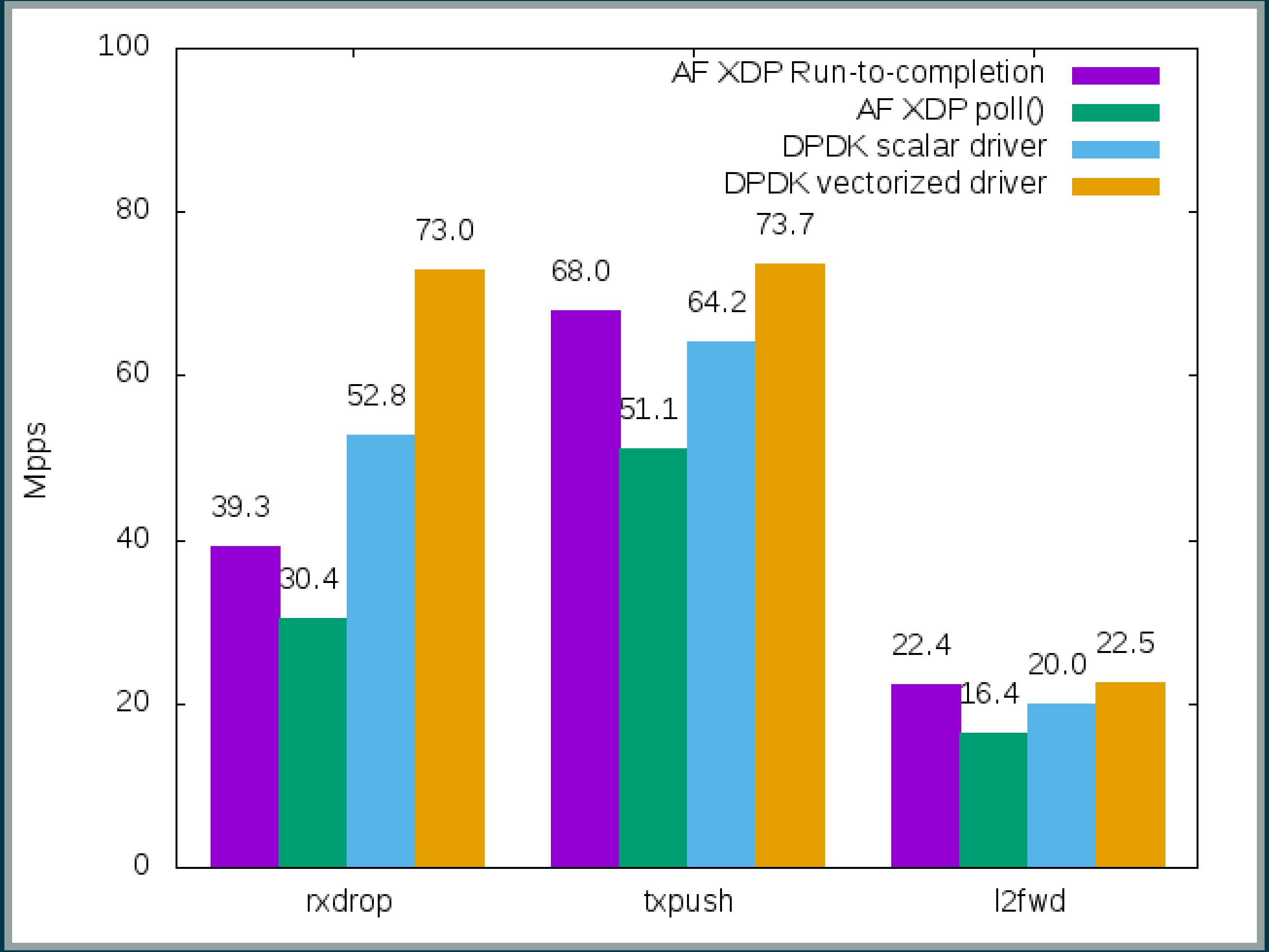
Two ways of running an AF_XDP application



Poll() Syscall Results



Comparison with DPDK



Integration with AF_XDP

How can kernel-bypass solutions use AF_XDP as a building block?

AF_XDP integration with DPDK

AF_XDP poll-mode driver for DPDK

- RFC patchset for AF_XDP PMD-driver sent on DPDK-mailing list by Intel
- ~1% overhead

Advantages:

- Don't monopolize entire NIC
- Split traffic to kernel with XDP filter program
- HW independent application binary
- Isolation and robustness
- Cloud-native support
- Fewer setup restrictions

AF_XDP integration with VPP

VPP (FD.io) could integrate via AF_XDP DPDK PMD

- But VPP uses only user-mode driver of DPDK
- VPP has a lot of native functionality

A native AF_XDP driver would be more efficient

- Less code and easier setup without DPDK

AF_XDP integration with Snabb Switch

Snabb Switch

- Implement an AF_XDP driver?
- Allow leveraging kernel drivers that implement XDP
  - Kernel community takes care of maintaining driver code
- Any performance loss/gap to native Snabb driver ?
  - E.g. NAPI “only” bulk up-to 64 packets
  - E.g. NAPI is not doing busy-polling 100%, more latency variance

Ongoing work

- Upstreaming performance optimizations
- XDP programs per queue
- Libbpf: facilitating adoption
- Packet clone for XDP

Summary

- XDP = Linux kernel fast path
- AF_XDP = packets to user space from XDP
- DPDK speeds
- A building block for a solution. Not a ready solution in itself.
- Many upcoming use cases,
  - e.g., OVS, XDP-offload netstack, DPDK PMD
- Come join the fun!
  - https://github.com/xdp-project/xdp-project

Backup Slides

Where does AF_XDP performance come from?

Lock-free channel directly from driver RX-queue into AF_XDP socket

- Single-Producer/Single-Consumer (SPSC) descriptor ring queues
- Single-Producer (SP) via bind to specific RX-queue id
  - NAPI-softirq assures only 1-CPU process 1-RX-queue id (per sched)
- Single-Consumer (SC) via 1-Application
- Bounded buffer pool (UMEM) allocated by userspace (register with kernel)
  - Descriptor(s) in ring(s) point into UMEM
  - No memory allocation, but return frames to UMEM in timely manner
- Transport signature Van Jacobson talked about
  - Replaced by XDP/eBPF program choosing to XDP_REDIRECT

Details: Actually four SPSC ring queues

AF_XDP socket: Has two rings: RX and TX

- Descriptor(s) in ring points into UMEM

UMEM consists of a number of equally sized chunks

- Has two rings: FILL ring and COMPLETION ring
- FILL ring: application gives kernel area to RX fill
- COMPLETION ring: kernel tells app TX is done for area (can be reused)

Gotcha by RX-queue id binding

AF_XDP bound to single RX-queue id (for SPSC performance reasons)

- NIC by default spreads flows with RSS-hashing over RX-queues
  - Traffic likely not hitting queue you expect
- You MUST configure NIC HW filters to steer to RX-queue id
  - Out of scope for XDP setup
  - Use ethtool or TC HW offloading for filter setup
- Alternative work-around
  - Create as many AF_XDP sockets as RXQs
  - Have userspace poll()/select on all sockets