redhat.

XDP – eXpress Data Path

An in-kernel network fast-path

A technology overview

Jesper Dangaard Brouer
Principal Engineer, Red Hat

TheCamp, Denmark
July 2017

Introduction

- This presentation is about XDP
    - Making people aware of this technology
    - Explaining the building blocks of XDP
        - Understand idea behind eBPF

What is the problem?

- Compared to bypass solutions, DPDK and netmap
  - Linux kernel networking is said to be slow
- Fundamental reason: Linux build on assumption
  - that most packets travel into sockets
    - takes cost upfront of allocating a "socket buff" (sk_buff/SKB)
- Linux lacks an in-kernel fast-path
  - DPDK bypass operate on "earlier" network layer
  - Kernel lack network layer before allocating SKBs

Why is an in-kernel fast-path needed?

- Today everything relies on networking
  - Kernel provides core foundation for network
- Solutions like DPDK: *make networking an add-on*
  - No longer part of core foundation everybody share
  - DPDK require maintaining full separate drivers
  - Special kernel boot parameters, 100% CPU usage
  - Harder to integrate into products/solutions
    - e.g. DPDK and containers are not compatible
    - e.g. you need to reimplement a TCP/IP stack
- Need in-kernel fast-path solution, part of core
  - that works in concert with the existing network stack

What is XDP (eXpress Data Path)?

- XDP is an in-kernel network fast-path facility
  - The "packet-page" idea from NetDev1.1 (Feb 2016) "rebranded"
  - Performance is primary focus and concern
    - Yes, it is as fast as DPDK and netmap
- XDP is NOT kernel bypass
  - Designed to work in concert with netstack
  - "Just" an earlier packet processing stage
  - Adaptive RX interrupt model (via NAPI)
- XDP run-time programmable: via eBPF
  - User-defined, sandboxed bytecode executed by the kernel

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: New building block for Networking

- XDP is a core kernel facility (since kernel v4.9)
  - Other Open Source projects pickup and use this
    - DDoS protection (PoC code for blacklist)
    - Cilium (Most promising and complete solution for container)
    - IOvisor/BCC - goal of creating userspace library
  - Companies already using XDP:
    - Facebook: DDoS + Load-balancer (10x boost vs. IPVS)
    - CloudFlare: DDoS protection (waiting for SolarFlare support)
    - One.com: DDoS protection

XDP introduce: earlier packet processing stage

- Traditionally Linux Kernel Networking
  - Rely on meta-data struct sk_buff (called "SKB")
    - Keep state and pointers to real packet-data
    - Assume most pkts reach deep into netstack (socket delivery)
  - Take alloc, setup and clear cost of SKB "upfront"
- XDP change this: “new layer in network stack”
  - Early parts of network stack don't need full SKB
    - XDP gives access to packet-data, before the SKB is allocated
  - As early as possible: hook in NIC drivers
  - Via programmable interface (eBPF)

Device driver dependency

- For high speed:
  - XDP depend on drivers implement RX hook
    - Luckily only software limitation
  - Fairly small change to drivers, low maintenance cost
    - especially compared to DPDK model of reimpl. drivers
- For ease of development: XDP "skb"-mode (v4.12)
  - Allow attaching XDP programs to any net_device
  - Makes it easier to devel and test XDP programs
  - Runs after SKB is allocated: obviously slower

Device drivers with Native XDP support

- Mellanox: mlx4 (v4.10) + mlx5 (v4.9)
- Netronome: nfp (v4.10)
- Virtio-net (v4.10)
- Cavium/Qlogic: qede (v4.10)
- Cavium: thunder/nicvf (v4.12)
- Broadcom: bnxt (v4.12)
- Intel: ixgbe (v4.12) + i40e (net-next)

What is eBPF? (**e**xtended **B**erkeley **P**acket **F**ilter)

- Originally programmable filter language for tcpdump
  - 1992 by Van Jacobson and Steven McCanne
- Alexei (3.18) generalized and extended instruction set
  - Introduced maps: key-value store, share-able
- eBPF: User-defined, sandboxed bytecode executed by the kernel
  - Lot of eBPF activity within tracing part of kernel
  - seccomp-bpf, filter syscalls (e.g. used by Docker, OpenSSH)
  - eBPF in Networking, many areas already
    - tcpdump + CPU steering, socket filter, iptables match module
    - Traffic-Control (tc) filter and actions for ingress/egress qdisc
      - Used by Cilium to speedup and secure container networking

XDP and eBPF user programmable networking

- 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
  - Customers can quickly implement something
    - Keeps policy choices outside kernel
      - Kernel is free from maintaining this forever \o/
  - Only run program code user actually need
    - No accumulative feature bloat
      - No need to run code everybody else *once* needed

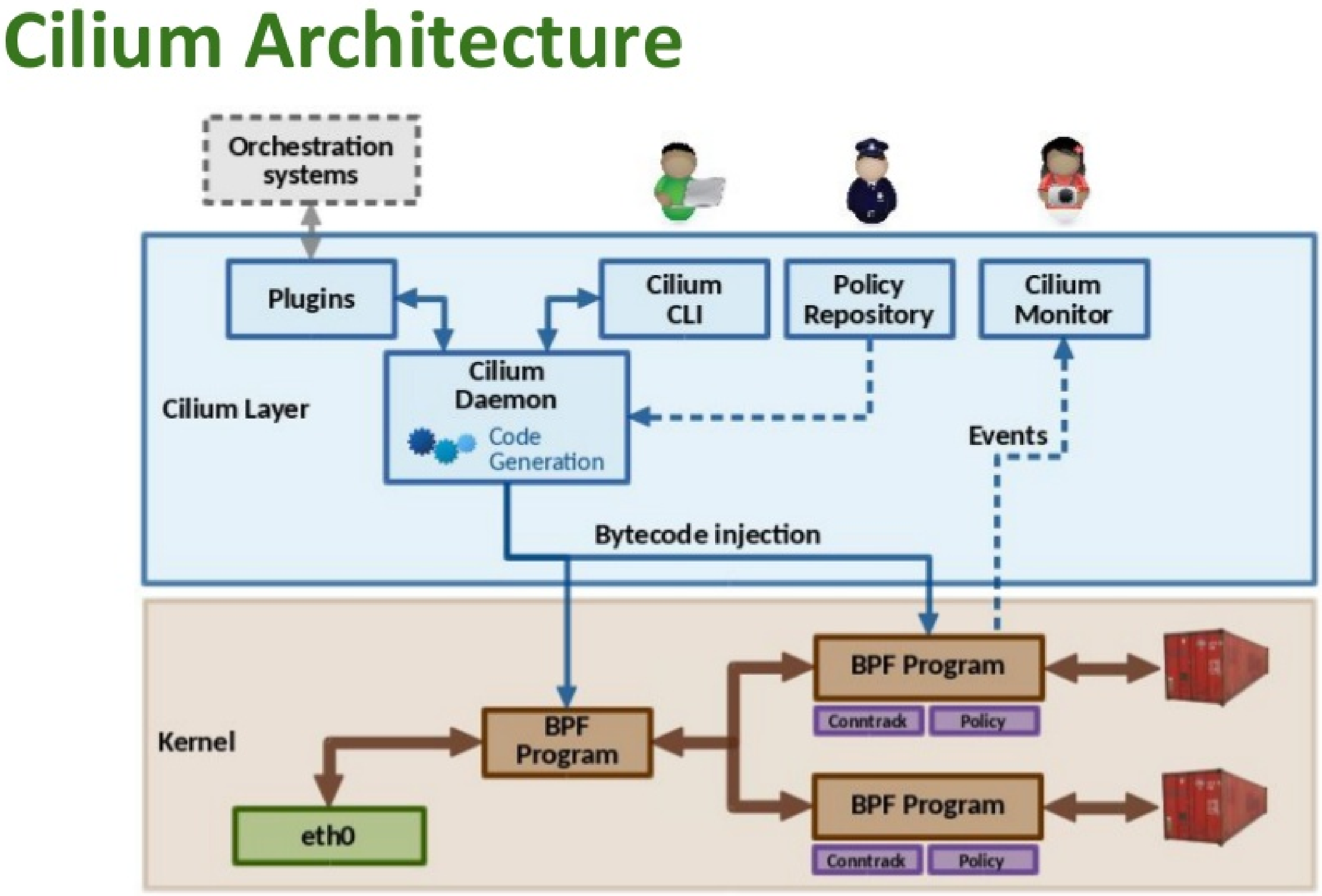
Fundamentally: XDP+eBPF gives adaptability

- XDP is also about maintainability and adapting quickly
- Customers want a long term stable kernel
  - but want to newest feature today
- XDP+eBPF gives programmable policies
  - Avoids creating kernel-ABI for every specific policy
  - Customer can adapt, without upgrading kernel
- Gives flexibility to adjust to the unknown
  - Cannot predict the future
    - instead add room for adapting quickly

eBPF as "micro-kernel" components

- Understand the architectural concept behind eBPF
  - eBPF program are not real programs
    - it is program snippets loaded into kernel
- See as components implementing specific behaviors
  - Glue them together
    - using maps, and **userspace orchestration**
- Building something that looks like “micro-services”
  - Just running inside the kernel
    - could see it as "micro-kernel" components

Credit: OVS conference 2016, Cilium architecture



eBPF programming model

- Restrictions: Run code in kernel in a safe environment
  - Must execute in short finite amount of time
  - Memory accesses strictly controlled and verified
  - No back branches allowed, and none needed due to MAPS
- eBPF byte-code is the assembler language
  - Full compilers exist from C and other languages into eBPF
  - Main compiler: LLVM, more interesting work coming
- eBPF maps are important
  - Adjust your programming mindset

eBPF maps: important core concept

- eBPF Maps, generic key-value store
  - Can store/keep state across invocation
  - Adjust programming mindset:
    - Can implement any Finite State Machine
      - see: token bucket https://github.com/qmonnet/tbpoc-bpf
- Can control bpf program flow via maps
  - Adjusting maps from userspace
- Maps can be shared between bpf programs
  - Exported via filesystem or file-descriptor
    - Easy privileged separation via file ownership permissions

eBPF - JIT (Just-In-Time) compiling

- How can eBPF byte-code be fast?
  - (Hint: it is not...)
- Kernel have JIT stage when loading eBPF
  - Transforms byte-code into
    - CPU native assembly instructions ←Hint: Very fast!
  - All 64-bit architectures are done
    - x86_64, arm64, ppc64, mips64, sparc64, s390x
- Smart-NICs looking at HW offloading eBPF
  - Like Netronome (driver nfp)

XDP core building blocks

- What can XDP do?
  - Can read and modify packet contents
  - Can push and pull headers
  - eBPF trigger actions based on return codes
    - **XDP_DROP** - very fast drop by recycling
      - DDoS mitigation
    - **XDP_PASS** – pass possibly modified packet to network stack
      - Handle and pop new unknown encap protocols
    - **XDP_TX** – Transmit packet back out same interface
      - Facebook use it for load-balancing, and DDoS scrubber
    - **XDP_ABORTED** – also drop, but indicate error condition
      - Tracepoint: xdp_exception
    - **XDP_REDIRECT** – Transmit out other NICs
      - Very new (est.4.14), (plan also use for steering packets CPUs + sockets)

Kickstarting XDP community

- Mailing list for newbies: xdp-newbies@vger.kernel.org
  - Up 3 month (since April 2017 NetDevConf 2.1): 233 emails
- Placed ready to use XDP code on github
  - prototype-kernel under samples/bpf/
  - Associated XDP/eBPF tutorial on YouTube
    - Given April 2017 at NetDevConf 2.1
- Started XDP doc project:
  - https://prototype-kernel.readthedocs.io
  - Cilium: “BPF and XDP Reference Guide”
    - http://cilium.readthedocs.io/en/latest/bpf/
- (p.s. eBPF have own community: iovisor-dev@lists.iovisor.org)

Future development and roadmap

- What are the missing features?
- What is on the roadmap?

Just merged action: XDP_REDIRECT

- New action return code: XDP_REDIRECT
  - Hope this will be last action code for drivers
  - Allow steering XDP packet buffer
- First obvious use: almost like XDP_TX
  - Transmit raw XDP packet out another NIC
    - (Delayed tailptr write, important for performance)
- Envision: Flexibility via bpf-maps
  - New redirect types, via adding new bpf map types

(Developing PoC code together with John Fastabend)

Roadmap: Redirect to remote CPUs (1/2)

- New type of redirect to remote CPU (Just a new map type)
- Problem it is trying to solve:
  - Slow (userspace) process on RX CPU cause bottleneck
  - Current solutions:
    - (1) RPS (Receive Packet Steering)
      - Happens after SKB alloc
      - Enqueue to remote CPU bottleneck
    - (2) Splitting workload via socket queue
      - Still bottleneck RX CPU
      - Too slow: e.g atomic mem-acct + queue management
      - SKB alloc and free happens on different CPUs

Roadmap: Redirect to remote CPUs (2/2)

- Solution: Transfer XDP packet to remote CPU
  - Alloc SKB on remote CPU
    - and free SKB likely on same CPU
  - RX bulk and bulk transfer is key for performance
    - page recycle pool facility needed for page performance
- Imply: building SKB outside driver
  - Interesting for driver simplification
  - Require: extra meta data to populate some SKB fields

Missing XDP/eBPF feature+helpers: RX hash

- XDP RX hash (have PoC code)
  - For correcting HW's hash to make RPS work
    - RPS = Receive Packet Steering
    - Seen issue with both VXLAN and Q-in-Q
      - Issue: NIC placed all packets on 1-CPU
  - Do XDP CPU redirect, based on flow hash
    - Without touching memory!
    - Basically faster version of RPS

Missing XDP/eBPF feature+helpers: XDP mark

- XDP mark transfer to SKB→mark (have PoC code)
  - Way of communicating between XDP and netstack
  - Trick used today:
    - XDP add VLAN header to packet steer to net_device
  - Alexei Starovoitov rejected first iteration
    - Want larger/generic "mark" value/area
      - Daniel Borkmann is working on this

Missing XDP/eBPF feature+helpers: csum

- XDP checksum helpers
  - Want csum bpf helpers (like for SKBs)
  - Make it easier to modify packets
    - Currently open-coded csum fixups in eBPF programs
  - Match on HW checksum validation info
    - Allow early drop on bad csums
  - Info from HW-desc needed later
    - When want to construct SKB outside driver

Missing feature negotiation

- Driver XDP hook challenge: bpf helper model
    - Core assume bpf helper code means feature avail
    - XDP driver might not implement feature
        - (like rxhash or mark)
        - Particular changes to xdp_buff are challenging
    - Unknown action codes not-critical
        - Just fall-through to XDP_ABORTED
- Patchset send as RFC
    - Top patch and followup patch

Supporting eBPF based solution

- What are the challenges and support cost
  - When customers start using eBPF?

- Sysadm perspective:
  - Customers want support and report issues
    - and might neglect to tell they are using eBPF
    - Sysadm need tools to “see” what is going on

Introspection into running eBPF progs

- Need tools for support purposes
  - Introspection into running eBPF programs
    - e.g basic listing of all running program
- eBPF program IDs (Got added very recently)
  - XDP export this ID
    - Can now identify what XDP program is running
- Expect better tools for
  - Extracting eBPF code from kernel
  - And better disassembly and objdump support

XDP tracepoints

- XDP have strategic tracepoints
  - Can be used for debugging exceptions
    - like XDP_ABORTED and XDP_TX failures
- Can also attach eBPF to tracepoints
  - Via maps, provide feedback loop to XDP program
    - particular when XDP_TX/redirect overflow target
- Likely new tracepoint for
  - XDP_REDIRECT to ease monitor forwarding

End slide: Summary

- The Linux Kernel needs an in-kernel fast-path
  - Bypass alternatives, is making networking an add-on
    - This is bad, networking need to be a core service
- XDP is the in-kernel fast-path solution
  - Part of and works in concert with existing network stack
  - Lower maintenance cost, as part of the Linux Kernel
  - New architecture for user programmable networking
    - Userspace in drivers seat
      - Via injecting "micro-kernel" components, solve specific needs

Extra slides

Product integration

- How does XDP relates to products?

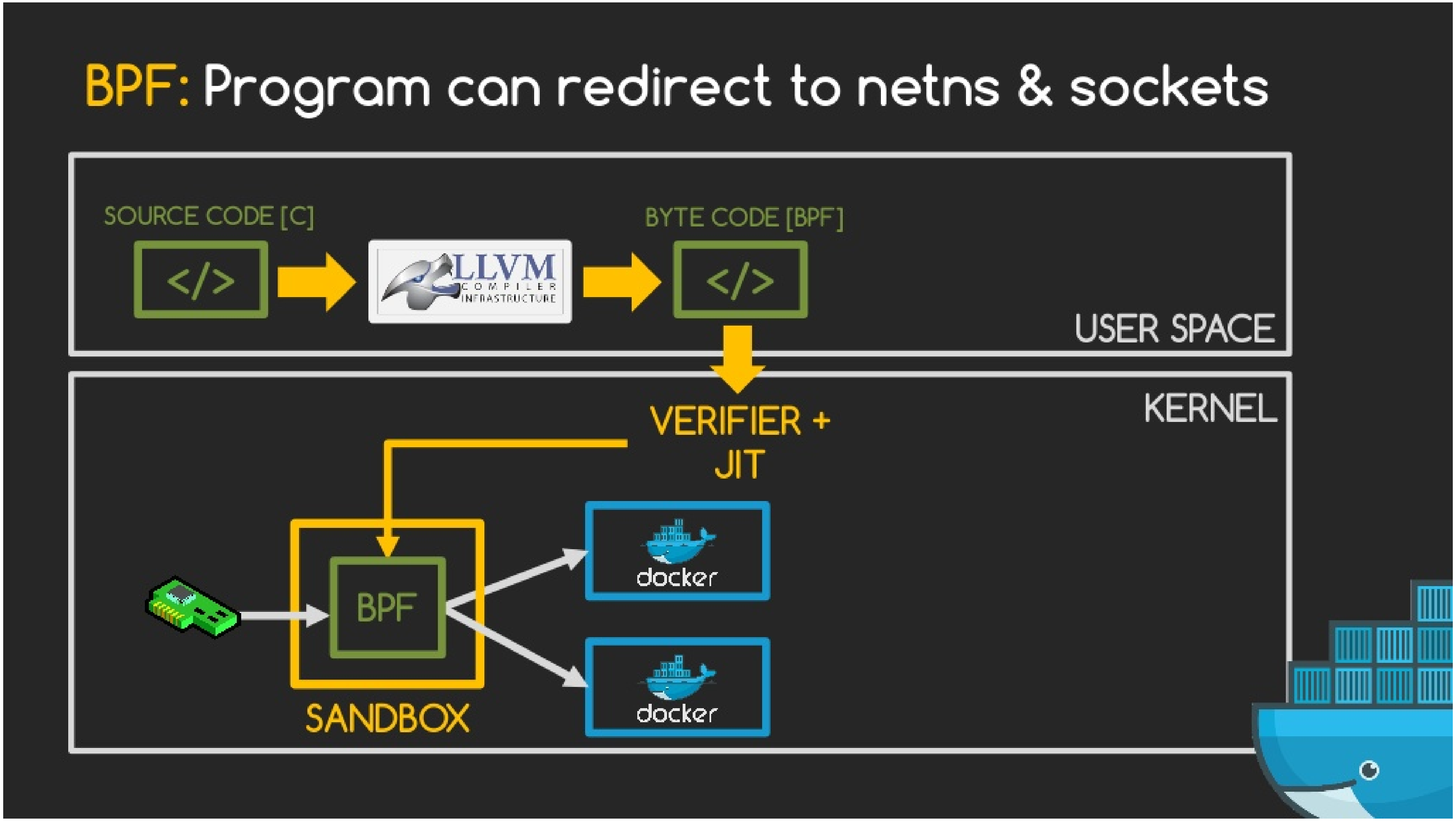
Product integration: VMs

- Products: OpenStack / OVS
- OpenStack Summit (Oct 2016, PlumGrid+Huawei) Video
  - Show using XDP for DDoS protection, protecting VMs
    - Drop inside VM cannot keep up
- With XDP_REDIRECT:
  - More direct delivery into VMs
  - SDN controller
    - Accelerate packet delivery via loading eBPF snippets

Product integration: Containers

- Products: OpenShift / Docker
- XDP can pop and rewrite IPs
  - Allows skipping some netstack layer
- With XDP_REDIRECT:
  - Skip netstack layers deliver directly to container veth
- eBPF with TC ingress and egress (avail today)
  - like Cilium
  - can already do more direct deliver into containers

Credit: DockerCon 2017 - Cilium


BPF: Program can redirect to netns & sockets

Product integration: Replace Network Appliances

- XDP programs on servers, instead of appliance
  - serve as 'bump in the wire' to
    - protect a rack of servers from DDoS attacks
    - or transparent load balancing
  - Drastically lower cost/Gbps than appliances
- RHEL supporting XDP enables
  - Companies develop these kind of boxes