# XDP – eXpress Data Path

## An in-kernel network fast-path

### A technology overview

Jesper Dangaard Brouer
Principal Kernel Engineer, Red Hat

Open Networking
Korea, Seoul
11th November 2017

# Introduction

- This presentation is about XDP
  - XDP – eXpress Data Path
    - Why Linux need it
  - Making people aware of this technology
    - XDP: New in-kernel building block for networking
  - Explaining the building blocks of XDP
    - Help 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. takes over entire NIC
    - 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



# 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)
    - RX cache prefetch loop trick
      - 20 Mpps XDP_DROP
    - TX-bounce with udp+mac rewrite:
      - 8.5Mpps – cache-miss
      - 12.3Mpps – RX prefetch loop trick



# 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 create userspace library (python)
  - Companies already using XDP:
    - Facebook: DDoS + Load-balancer (10x boost vs. IPVS)
    - CloudFlare: DDoS protection (waiting for SolarFlare support)
    - One.com: DDoS protection



# 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
  - XDP_**TX** – Transmit packet back out same interface
  - XDP_**ABORTED** – also drop, but indicate error via tracepoint
  - XDP_**REDIRECT** – Transmit out other NICs or steer
- All BPF programs can also interact via
  - Call helper function that lookup or modify kernel state
  - Create state via shared maps (both userspace and other bpf-progs)



# XDP **DDoS Open Source Community,** PLEASE

- The kernel tech is ready, start community project
  - Trust me, it's better to share your toys
- Point with XDP+BPF technology
  - You can quickly adapt BPF code-snippets
    - Hopefully quicker than attacker
    - Don't wait for a vendor to update your protection-software
- Need (shared) pool of DDoS mitigation programs
  - The more examples the better
  - Allow up to adapt and deploy counter measures
  - Be a community fighting DDoS



# 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 + 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
  - Administrators can quickly implement something
    - No need to upgrade kernel
  - Only run program code needed for use-case
    - No accumulative feature bloat



# 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



# Example: Adapting to unknown protocol

- How to handle new protocol/encapsulation
  - That the kernel doesn't know yet?
    - Without upgrading the running kernel!
- On RX: XDP adjust packet headers
  - to something kernel understand
    - E.g. steer into VLAN devices
- On TX: BPF can add back (encapsulation) headers
  - With BPF hooks in Traffic Control or Socket filter
    - Restore info, based on shared BPF-map
    - Or based on VLAN device or SKB-marking

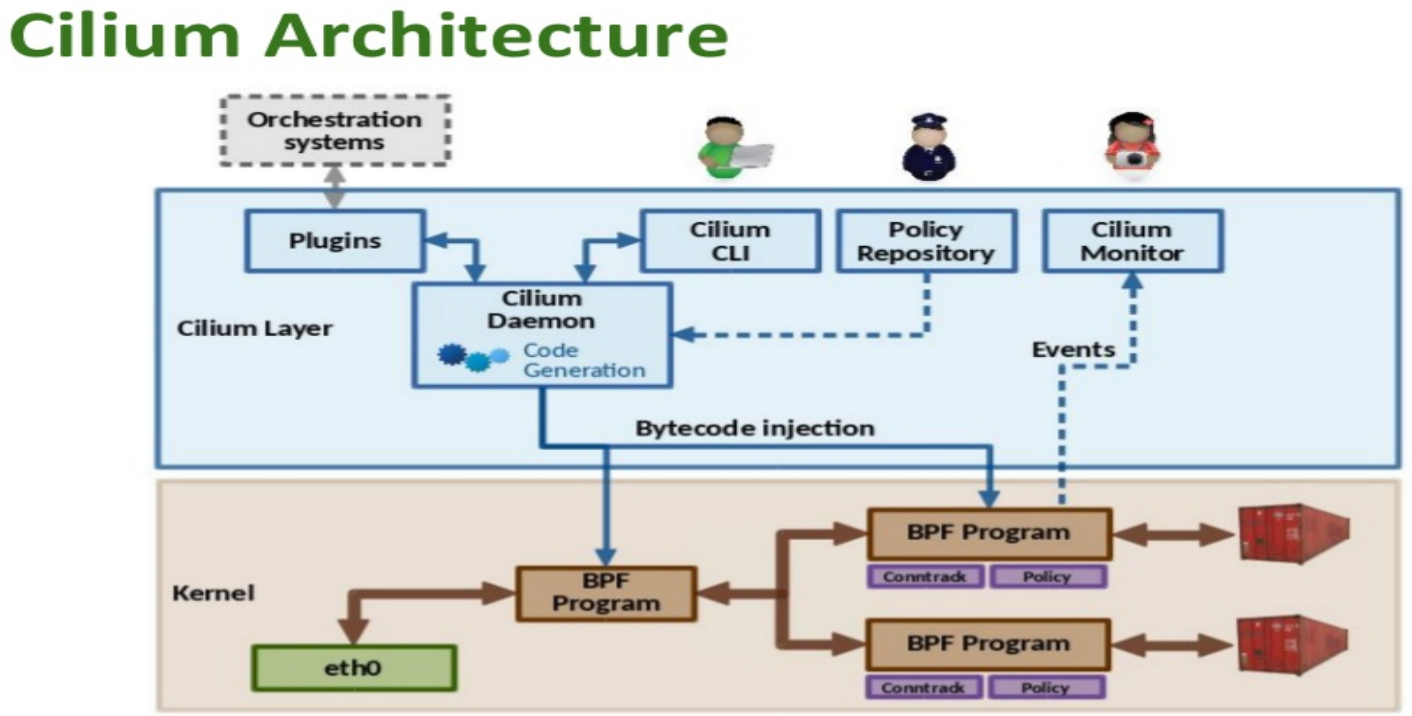


# eBPF as "micro-kernel" components?

- Real power comes from using multiple BPF-hooks
- 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 loop 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
    - Setup of build env needed, distro support since Fedora 25
- 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, file-descriptor or map-id
    - 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)



# New action: XDP_REDIRECT

- New action return code: XDP_REDIRECT
  - Innovative part: Redirect using maps (use bpf_redirect_map())
- Redirect via maps: RX bulking, via flush operation after napi_poll
  - Dynamic adaptive bulking
    - Method of adding bulking without introducing additional latency
    - Bulk only frames available in driver NAPI poll loop
- New map types for redirect
  - devmap - BPF_MAP_TYPE_DEVMAP
    - Bulk effect via delaying HW tail/doorbell (like xmit_more)
  - cpumap - BPF_MAP_TYPE_CPUMAP
    - Bulk 8 frame to remote CPU, amortize cross CPU cost
    - Provide CPU separation at XDP “layer”



# Introspection into running eBPF progs

- eBPF program IDs (kernel v4.13)
  - XDP export this ID
    - Can now identify what XDP program is running
- Tool under development named: `bpftool`
  - Part of Kernel tree: tools/bpf/bpftool/
  - Allows inspection and simple modify BPF objects
  - Easy to list all programs currently loaded
  - Support output in JSON format



# The XDP community

- Mailing list for newbies: xdp-newbies@vger.kernel.org
  - ~6 month (since April 2017 NetDevConf 2.1): 391 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 )



# 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
- Tracepoints exists for
  - XDP_REDIRECT to ease monitor forwarding
    - Sample tool avail (v4.14): xdp_monitor



# End slide: Summary

- The Linux Kernel got new in-kernel fast-path
  - Bypass alternatives is making networking an add-on
    - This is bad, networking need to be a core service
- XDP is an 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
    - Users in driver’s seat via BPF snippets
  - Don’t take over entire NIC
    - BPF program as programmable filter



# Thanks to

- XDP + BPF combined effort of many people
  - Alexei Starovoitov (Facebook)
  - Daniel Borkmann (Covalent)
  - Brenden Blanco (Vmware)
  - Tom Herbert (Quantonium, former Facebook/Google)
  - John Fastabend (Covalent, former Intel)
  - Martin KaFai Lau (Facebook)
  - Jakub Kicinski (Netronome)
  - Michael S. Tsirkin (Red Hat)
  - Jason Wang (Red Hat)
  - Saeed Mahameed (Mellanox)
  - Tariq Toukan (Mellanox)
  - Edward Cree (Solarflare)



# 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

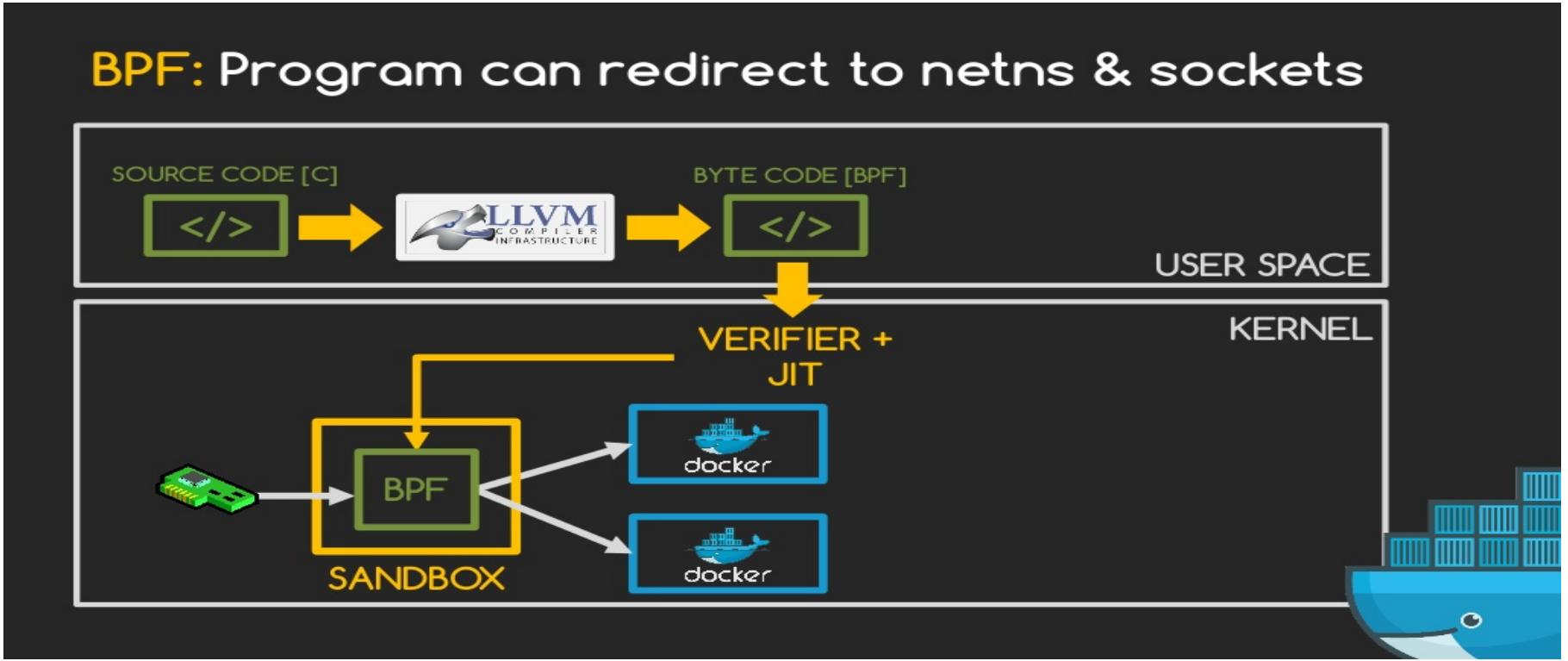


# Product integration: Containers

- 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



# 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