eBPF and XDP walkthrough and recent updates

Daniel Borkmann
<daniel@iogearbox.net>
cilium project

fosdem17, February 4, 2017
Big Picture: eBPF and Networking

- eBPF: efficient, generic in-kernel bytecode engine
- Today used mainly in networking, tracing, sandboxing
  - XDP, tc, socket reuseport/demux/filter, perf, bcc, seccomp, ...
- cls_bpf programmable packet processor in tc subsystem
  - Attachable to ingress, egress of kernel’s networking data path
- XDP programmable, high-performance, in-kernel packet processor
  - Attachable to ingress directly at driver’s early receive path
- cls_bpf complementary to XDP
  - Attachable on ingress and egress to all net devices
  - skb as input context to leverage stack functionality
eBPF Architecture

- 11 64bit registers, 32bit subregisters, stack, pc
- Instructions 64bit wide, max 4096 instructions/program
- Various new instructions over cBPF
- Core components of architecture
  - Read/write access to context
  - Helper function concept
  - Maps, arbitrary sharing
  - Tail calls
  - Object pinning
  - cBPF to eBPF translator
  - LLVM eBPF backend
- eBPF JIT backends implemented by archs
- Management via bpf(2), stable ABI
tc’s cls_bpf and sch_clsact

- **sch_clsact** container for tc classifier and actions
- Provides two central hooks in data path
  - **Ingress:** __netif_receive_skb_core()__
  - **Egress:** __dev_queue_xmit()__
- **cls_bpf** runs eBPF, allows for atomic updates
- Fast-path with direct-action (da) mode
  - Verdicts: ok, shot, stolen, redirect
- Offload interface implementable by drivers: nfp
- C → LLVM → eBPF → ELF → tc → verifier → JIT → cls_bpf → offload
XDP (eXpress Data Path)

- Objectives and use-cases
  - Generic framework for high-performance packet processing
  - Runs eBPF program in driver at earliest possible point
  - Works in concert with the kernel (same security model, no out-of-tree)
  - Packet stays in kernel, no need for crossing boundaries
  - DSR load balancing, forwarding, anti DDoS, firewalls, monitoring
  - Verdicts: aborted, drop, pass, tx

- Currently supported: mlx4, mlx5, nfp, qed, virtio_net, i40e*, bnxt*
- Allows for atomic updates (currently driver dependent)
- Offload interface implementable by drivers: nfp

C → LLVM → eBPF → ELF → ip → verifier → JIT → XDP → offload

*: merge expected soon, patches posted on netdev

user space, kernel space
## XDP and cls_bpf Features

- **Generic maps (lookup, update, delete):**
  - Array map*
  - Hash table*
  - LRU map*
  - LPM trie

- **Specialized maps (used with helpers):**
  - Program array
  - Perf event map
  - Cgroups v2 map

- **Packet access:**
  - Direct packet read
  - Direct packet write
  - Additional metadata in context
  - Metadata mangling (proto, type, mark, etc)

---

*: also as per-CPU and preallocated map flavor  
†: not yet seen by stack
XDP and cls_bpf Features

- Packet forwarding:  
  TX to same port ✓ ✓  
  TX to any netdevice (including virtual) ✓ *  
  TX to RX ✓  

- Miscellaneous:  
  Encapsulation ✓ † ✓  
  Headroom mangling ✓  
  Tailroom mangling ✓  
  Event notification (including payload) ✓ ✓  
  Tail calls ✓ ✓  
  Checksum mangling ✓ ✓  
  Packet cloning ✓  
  Cgroups v1/v2 ✓  
  Routing realms ✓  
  ktime, CPU/NUMA id, rand, trace printk ✓ ✓

*: mid/long-term for multiport and different physical device  
†: restricted to collect metadata, f.e. vxlan, geneve, gre, ipip, etc.
iproute2 as eBPF loader

- Frontend for loading networking eBPF programs into kernel
- Shared backend library for ELF loader
- Map relocation, tail call and object pinning handling
- cls_bpf workflow:
  
  ```
  $ clang -O2 -target bpf -o foo.o -c foo.c
  # tc qdisc add dev em1 clsact
  # tc filter add dev em1 ingress bpf da obj foo.o sec p1
  # tc filter add dev em1 egress  bpf da obj foo.o sec p2
  # tc filter del dev em1 ingress
  # tc filter del dev em1 egress
  # tc qdisc del dev em1 clsact
  ```

- XDP workflow:
  
  ```
  $ clang -O2 -target bpf -o foo.o -c foo.c
  # ip [-force] link set dev em1 xdp obj foo.o
  # ip link set dev em1 xdp off
  ```
JITs, Offload, Hardening

- Available as of today: x86_64, arm64, ppc64, s390x
  - net.core.bpf_jit_enable=1
  - ppc64: initial JIT merged and tail call support added
  - arm64: tail call support, various optimizations, xadd still missing

- Offloading of eBPF to NIC via JIT: nfp

- Various hardening measures done by default, f.e. read-only marking

- Constant blinding infrastructure
  - net.core.bpf_jit_harden=1
  - Blinding for non-root programs enabled
  - Rewriting 32/64bit constants generically at BPF instruction level
  - \( \text{imm} \rightarrow ((\text{rnd} \oplus \text{imm}) \oplus \text{rnd}), \text{ins}_{\text{imm}} \rightarrow \text{ins}_{\text{reg}} \)
Other Recent Improvements

- DWARF support for LLVM eBPF backend
- Various verifier improvements wrt LLVM code generation
- Dynamic map value and stack access
- eBPF hooks for lightweight tunneling and per cgroups v2
- Tracepoint infrastructure for eBPF and XDP
- eBPF verifier and map selftest suite
- kallsym support for JIT images (to be submitted soon)
Thanks!

- Couple of next steps
  - Verifier improvements (e.g. logging, pruning)
  - Widespread XDP support, improved forwarding
  - Better map memory management
  - Inline map lookup, bounded loops, etc

- Code
  - cilium project: github.com/cilium
    - BPF & XDP for containers
  - git.kernel.org → kernel, iproute2 tree

- Further information
  - netdev conference proceedings
  - Kernel tree: Documentation/networking/filter.txt
  - qmonnet.github.io/whirl-offload/2016/09/01/dive-into-bpf