FOSDEM’20 • Brussels, 2020-02-01

Endless Network Programming
•
An Update from eBPF Land

Quentin Monnet
@qeole

Outline

- eBPF Basics
- New Features
- eBPF Universe

eBPF Basics

BPF Architecture

extended Berkeley Packet Filter

- Programs compiled from C (or Go, Rust, Lua): clang/LLVM backend
- `bpf()` syscall to inject into the kernel
- Verifier for safety and termination
- JIT (Just-In-Time) compiling (optional)
- Programs attached to a hook in kernel (socket, TC, XDP, kprobes...)

Characteristics:

- 64 bit instructions
- 11 registers
- 512 B stack
- Up to 4096 instructions (or up to 131,072 simulated by the verifier)

- No loops allowed

BPF Architecture

extended Berkeley Packet Filter

- Programs compiled from C (or Go, Rust, Lua): clang/LLVM backend
- `bpf()` syscall to inject into the kernel
- Verifier for safety and termination
- JIT (Just-In-Time) compiling (optional)
- Programs attached to a hook in kernel (socket, TC, XDP, kprobes...)

Characteristics:

- 64 bit instructions
- 11 registers
- 512 B stack (→ but up to 1024 B with extension program)
- Up to 4096 instructions (or up to 131,072 simulated by the verifier)
  → Root: up to 1 million simulated instructions (v5.2)
- No loops allowed → Bounded loops (v5.3)

Performance Improvements

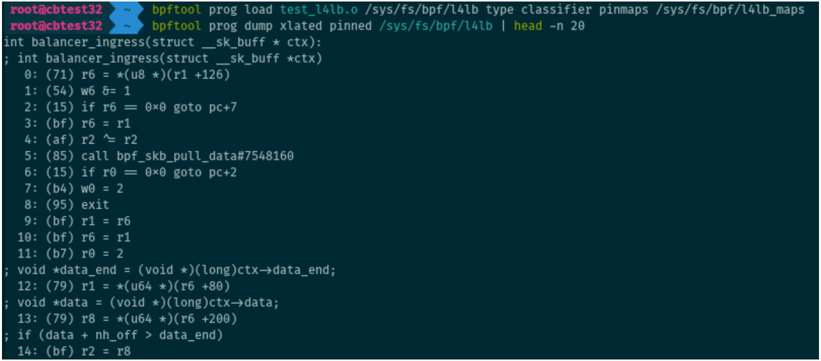
Many performance improvements, for example:

- LLVM can favour 32-bit subregisters
  Improved JIT efficency for 32-bit instructions on some architectures (up to 40% fewer instructions) (v5.3)
- Batched map operations via new BPF commands for maps (v5.6)
  Allow for faster processing
  No need to cycle on entries, no risk to hit a deleted entry
  - `BPF_MAP_LOOKUP_BATCH`
  - `BPF_MAP_LOOKUP_AND_DELETE_BATCH`
  - `BPF_MAP_UPDATE_BATCH`
  - `BPF_MAP_DELETE_BATCH`
- AF_XDP gets some love, too

New Features

BTF: *BPF Type Format*

Close to DWARF, provides debug information for BPF programs and maps
E.g. Source code in C for BPF program:



```
root@cbtest32 ~ bpftool prog load test_l4lb.o /sys/fs/bpf/l4lb type classifier pinmaps /sys/fs/bpf/l4lb_maps
root@cbtest32 ~ bpftool prog dump xlated pinned /sys/fs/bpf/l4lb | head -n 20
int balancer_ingress(struct __sk_buff * ctx):
; int balancer_ingress(struct __sk_buff *ctx)
   0: (71) r6 = *(u8 *)(r1 +126)
   1: (54) w6 &= 1
   2: (15) if r6 == 0x0 goto pc+7
   3: (bf) r6 = r1
   4: (af) r2 ^= r2
   5: (85) call bpf_skb_pull_data#7548160
   6: (15) if r0 == 0x0 goto pc+2
   7: (b4) w0 = 2
   8: (95) exit
   9: (bf) r1 = r6
  10: (bf) r6 = r1
  11: (b7) r0 = 2
; void *data_end = (void *)(long)ctx->data_end;
  12: (79) r1 = *(u64 *)(r6 +80)
; void *data = (void *)(long)ctx->data;
  13: (79) r8 = *(u64 *)(r6 +200)
; if (data + nh_off > data_end)
  14: (bf) r2 = r8
```

BTF: *BPF Type Format*

- Has been around since v4.18, but evolving a lot
- Generated by pahole or LLVM, verified in the kernel
- Kernel data embedded as BTF
  - Needs `CONFIG_DEBUG_INFO_BTF=y`
  - BTF data at /sys/kernel/btf/vmlinux
  - Used to access struct fields directly, instead of (fragile) offset
- Necessary for CO-RE (*Compile Once, Run Everywhere*), for tracing mostly
- More and more features rely on it internally

Global Data

- Global data support in C sources (v5.2)
- Global variables in .data, .rodata, .bss sections
  Templating: Just update contents in those sections in object file
- Global data can be `mmap()`'ed for easier access (v5.5)
- Close to global data: external variables (v5.6)
  (`LINUX_KERNEL_VERSION` and `CONFIG_XXX`)

BPF Trampoline

- Converts native calling convention into BPF calling convention (v5.5)
- New way to attach BPF programs to k(ret)probes: `fentry`, `fexit`
  Nearly zero overhead
- Such `fentry`/`fexit` programs can be attached to entry/exit of any networking BPF program: see input and output packets for TC, XDP etc.
- *BPF dispatcher*: Reuse trampoline to avoid retpoline cost for XDP programs (v5.6)

Global Functions, Dynamic Linking

- Global (non-`static`) functions supported by libbpf (v5.5)
- Dynamic program extensions (v5.6)
  New program type: `BPF_PROG_TYPE_EXT`, can dynamically replace a placeholder global function
- Advantages:
  - Dynamic policies
  - Code reuse
  - Shorter verification time

BPF STRUCT_OPS

- Overwrite `struct ops` in kernel with BPF programs
- New program/map types:
  `BPF_PROG_TYPE_STRUCT_OPS`, `BPF_MAP_TYPE_STRUCT_OPS`
- Example: `struct tcp_congestion_ops` can be replaced to implement custom TCP congestion control (e.g. from DCTCP)
- The `struct ops` to replace need some wrapping in the kernel, though

More to Come!

Developers in the community working on:

- XDP improvements
  - Multi-buffer (jumbo-frames, packet header split, TSO/LRO)
  - egress XDP
- Static linking (several object files merged into single program)
- Step-by-step debugging
- Not-networking use cases: LSM (Linux Security Module)

eBPF Universe

Tools and Projects

- bpftool / libbpf
  - Support for BTF
  - Generally: support for all new BPF features
  - Can generate “skeleton” header from object file, very helpful for working (and `mmap()`’ing) global data
- Katran (anti-DDoS, Facebook), Suricata (IDS), anti-DDoS (Cloudflare), etc.
- Cilium: Many new features (see next presentation!)
  Network, service and security observability tool: Hubble
- Tracing: Rezolus (Twitter), Sysdig, etc.
- “BPF as universal dataplane” project by big network players, early stage

Wrapping Up

- BPF development extremely active
- New features, new use cases (and that was just for networking)
- More to come!

Thank you!

Questions?