Comprehensive BPF offload

Nic Viljoen & Jakub Kicinski
Netronome Systems

netdev 2.2, Seoul, South Korea
November 8-10, 2017

An NFP based NIC (1U)
• Refresher
  • Programming Model
  • Architecture
• Performance & Optimization
• Requirements for Production Offload
  • bpftool
  • Verifier restructuring
• Program is written in standard manner

• LLVM compiled as normal

• iproute/tc loads the program requesting offload

• The `nfp_bpf_jit.c` converts the eBPF bytecode to NFP machine code (and we mean the actual machine code :))

• Translation reuses a significant amount of verifier infrastructure
Refresher-BPF Offload Mapping

1. 10 Registers (64 bit, 32 bit subregisters)
2. 512 byte stack
3. Maps, varying sizes
4. Driver
5. GPRs
6. LMEM (1 KB)
7. Thread (x4 per Core)
8. Core (x60 used for BPF)
9. CLS (64 KB)
10. CTM (256 KB)
11. Island (x6 per Chip)
12. IMEM (4 MB)
13. Chip
14. DRAM (2 GB)
Performance

- Simple XDP load balancer (~ 800 BPF insns, 4 lookups)
  - Based on the TC example in kernel - `selftests/bpf/l4lb.c`
  - Combined with `samples/bpf/xdp_tx_iptunnel_kern.c`
- Per CPU array changed to standard array to run offloaded
  - There is no nice equivalent for per CPU at the moment on the NIC
- Not optimised-big health warning :)
Future Optimizations

- Map placement/caching-as shown on previous page
- Using Packet Cache-reduce latency of packet accesses from ~50 cycles to ~3 cycles
- 32 bit ALU from LLVM where possible-reduce ALUs from ~ 4 machine code insns to 1
- Remove FW locks-double memory bandwidth
Requirements For Production Readiness

Multi-stage processing:
- Reliable manner to run some programs in host if not possible/desirable in offload

Debug:
- Usable verifier error messages
- Introspection - both of maps and programs

JIT:
- Translation before optimization
Multi-Stage Processing

- Offload some programs
  - This can be managed by the driver (implicitly) or explicitly
  - Use `data_meta` to inject programs into the correct BPF program to run next
    - Important for edge case where the next program to run is not fixed
  - Allows offload to be used for beneficial cases only
    - Can be explicitly via flags
Progress made - kernel

Upstream:

- new instructions (Daniel, Jiong, I);
- direct packet access;
- stack support;
- adjust head helper;
- add 32-bit subregister support to LLVM (Jiong).

Prototyped/PoC:

- map offload support (hash and array maps);
- atomic add operation;
- memcpy optimizations (Jiong);
- initiate work on register state tracking (Jiong).
Progress made - tooling

bpftool

- in kernel tree for Linux 4.15 (in the `tools/` directory);
- iproute2-like syntax;
- list and pin objects;
- programs:
  - show type, name, tag, id, memory usage, load time, used maps;
  - dump JITed and translated images (to file or print instructions);
- maps:
  - show type, name, id, key/value size, number of elements, flags;
  - lookup, update, delete, etc.
- JSON output (Quentin);
- BPF FS integration (Quentin, Prashant).
llvm-mc (Jiong)

- upstream for LLVM 6.0;
- LLVM’s macro assembler;
- verifier-style syntax:
  ```
  r1 = r6
  r2 = 0xff000000 ll
  call 12
  r0 = 0
  exit
  ```
- allows hand-crafting precise BPF programs (or compiling C code into assembly and modifying it);
- opens way for BPF inline assembly;
- very useful for testing particular instruction sequences.
Kernel basics (*refresher*)

- user space
- kernel space

**BPF syscall**
- program
- type (sk filter, kprobe, cls, xdp)
- license
- ...

**verifier**
- verification
- modification

**BPF prog**

**tc**
- fd
- fd, skip_* flags

**TC**
- cls_bpf

**offload object**

**NDP setup tc**

**stats & maps**

**RX**
**XDP**
**TX**

**HW JIT / translator**

**host JIT**

**fd, skip_* flags**

**controller**

**object**
- fd, skip_* flags

**driver**
Translation and loading (refresher)

- program
- type (sk filter, kprobe, cls, xdp)
- license
- ...

Verifier

- verify

User space

Kernel space

- fd
- fd, skip_* flags
- fd, skip_* flags

Verifier

3) Collect state/analyze

4) Optimize
5) JIT/generate image
6) Load image

(2) Re-run the verifier

(1) Check HW capabilities and image parameters

Hardware JIT / translator

ndo setup tc

XDP

Driver

Kernel

Translation and loading (refresher)

- program
- type (sk filter, kprobe, cls, xdp)
- license
- ...

Verifier

- verify

User space

Kernel space

- fd
- fd, skip_* flags
- fd, skip_* flags

Verifier

3) Collect state/analyze

4) Optimize
5) JIT/generate image
6) Load image

(2) Re-run the verifier

(1) Check HW capabilities and image parameters

Hardware JIT / translator

ndo setup tc

XDP

Driver

Kernel
Kernel basics (refresher)

- program
- type (sk filter, kprobe, cls, xdp)
- license
- ...

- verifier
- BPF prog
- host JIT
- modification
- verification

- BPF syscall
- fd
- fd, skip_* flags
- fd, skip_* flags

- tc
- TC
- cls_bpf
- offload object

- HW JIT / translator
- ndo setup tc
- stats & maps

- RX
- XDP
- TX
- driver
Progress made - kernel

- program
- type (sk filter, kprobe, cls, xdp)
- license
- ifindex

BPF syscall

- find device
- BPF prog
- offload

- verification
- modification

- JIT

driver ops

- XDP
- verifier prep
- translate
- destroy

offload object

RX XDP TX

ndo setup tc

driver

XDP ctrl

tc

cls_bpf

JIT

modification

verification

find device

user space

kernel space
Rationale for recent kernel changes

- allow device translator to access the loaded program as-is:
  - IDs/offsets not translated:
    - structure field offsets;
    - functions;
    - map IDs.
  - no prolog/epilogue injected;
  - no optimizations made;
- output errors at program load and map creation time;
- make use of access to the verifier log;
- include device information in introspection APIs (bpftool);
- dump translated image:
  - similar to host “JITed image”;
  - BPF core already has access to offload state (no longer driver black box);
- need to report machine info?
Debug and tooling APIs

- netlink extack support in cls_bpf/TC offloads:
  - XDP already carries extack for use by the drivers;
  - allows easier error reporting at attachment time;
- bpf perf event output - output samples for debugging the datapath;
- simple API for enabling/disabling optimizations:
  - verifier/kernel already has some simple optimizations (e.g. lookup inlining);
  - nfp translator already has a few and we expect to add more;
  - need to report, enable/disable optimizations with nice granularity;
- maps:
  - create maps on the device from the start;
  - simplify map load/eviction and locking greatly;
  - report errors/resource exhaustion at map creation time.
The end

Thank you!