A Primer on eBPF 

(Or, *'WebAssembly for the Linux Kernel'*)

Kyle A. Simpson
ks@arista.com
ksArista https://mcfelix.me

EOS+ Tech Talk – 13th April, 2023

Arista Networks

ARISTA

eBPF: What?

Figure 1: *'eBee', the eBPF Mascot.*

- *Simple* register machine VM bytecode for user-written code.

eBPF: What?

Figure 1: *'eBee', the eBPF Mascot.*

- *Simple* register machine VM bytecode for user-written code.
- Attach logic to *syscalls* & *hooks*.

eBPF: What?

Figure 1: *'eBee', the eBPF Mascot.*

- *Simple* register machine VM bytecode for user-written code.
- Attach logic to *syscalls* & *hooks*.
- (Somewhat) common compile target 🦀.

eBPF: What?

Figure 1: *'eBee', the eBPF Mascot.*

- *Simple* register machine VM bytecode for user-written code.
- Attach logic to *syscalls* & *hooks*.
- (Somewhat) common compile target 🦀.
- ...and very exciting for host networking via AF_XDP!

eBPF: What?

Figure 1: *'eBee', the eBPF Mascot.*

- *Simple* register machine VM bytecode for user-written code.
- Attach logic to *syscalls* & *hooks*.
- (Somewhat) common compile target 🦀.
- ...and very exciting for host networking via AF_XDP!
- Windows support now, too!

eBPF: Why?

- Lightweight instrumentation and debugging of:

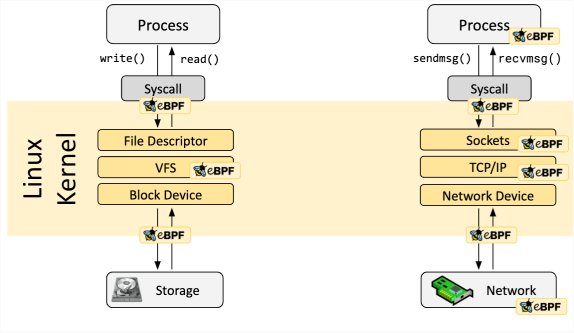


Figure 2: *eBPF Hook points.*

eBPF: Why?

- Lightweight instrumentation and debugging of:
  - the network stack,

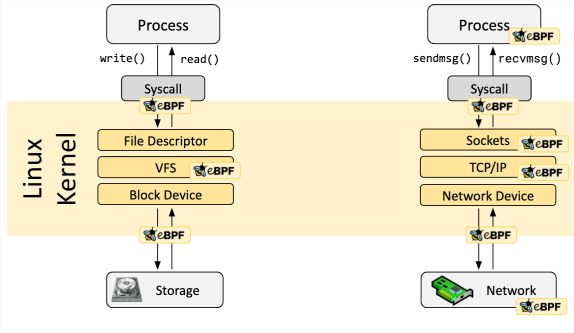


Figure 2: *eBPF Hook points.*

eBPF: Why?

- Lightweight instrumentation and debugging of:
  - the network stack,
  - the file system,

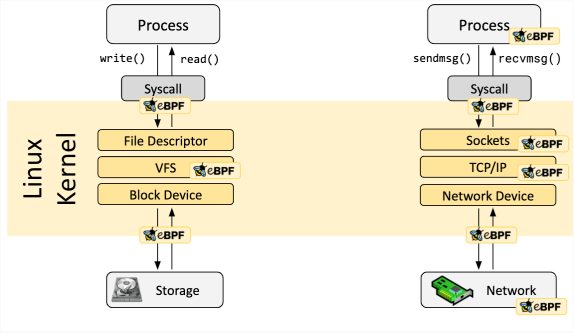


Figure 2: *eBPF Hook points.*

eBPF: Why?

- Lightweight instrumentation and debugging of:
  - the network stack,
  - the file system,
  - kernel functions,

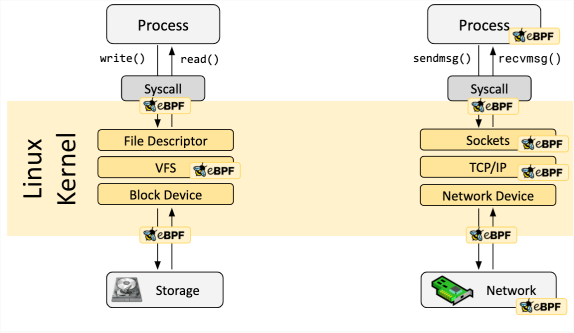


Figure 2: *eBPF Hook points.*

eBPF: Why?

- Lightweight instrumentation and debugging of:
  - the network stack,
  - the file system,
  - kernel functions,
  - drivers and hardware...

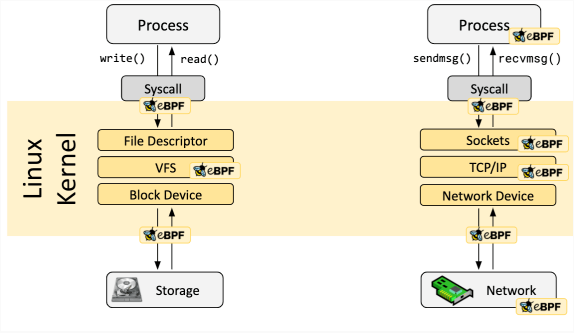


Figure 2: *eBPF Hook points.*

eBPF: Why?

- Lightweight instrumentation and debugging of:
  - the network stack,
  - the file system,
  - kernel functions,
  - drivers and hardware...
- *Network stack programmability*.

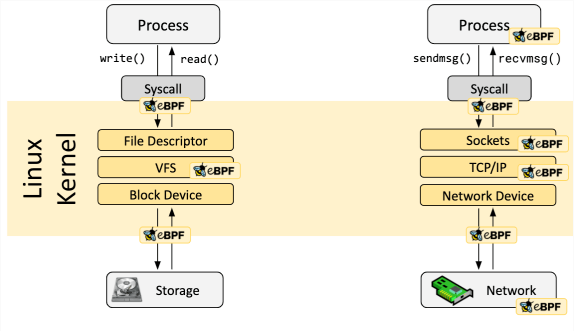


Figure 2: *eBPF Hook points.*

eBPF: Why?

- Lightweight instrumentation and debugging of:
  - the network stack,
  - the file system,
  - kernel functions,
  - drivers and hardware...
- *Network stack programmability*.
- JIT compiled (*x86_64*, *AArch64*).

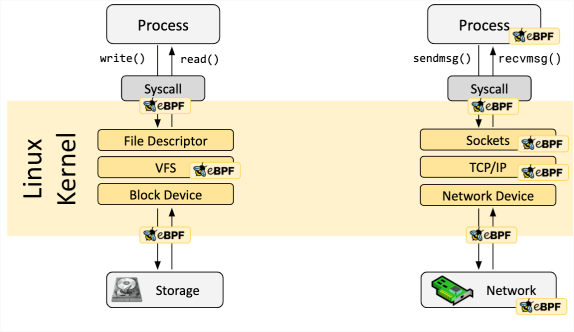


Figure 2: *eBPF Hook points.*

eBPF: Why?

- Lightweight instrumentation and debugging of:
  - the network stack,
  - the file system,
  - kernel functions,
  - drivers and hardware...
- *Network stack programmability*.
- JIT compiled (*x86_64*, *AArch64*).
- Kernel-verified and sanitised – secure & safe.

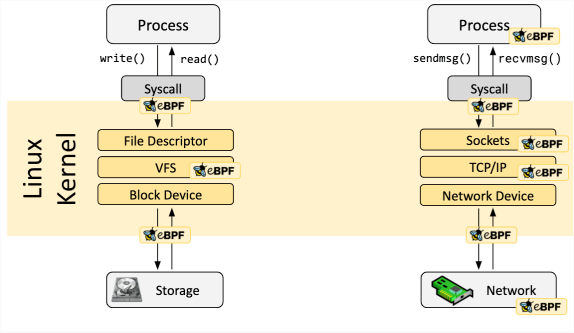


Figure 2: *eBPF Hook points.*

Who's using it?

Cloudflare DDoS attack scrubbing (*flowtrackd*[1]).

[1] Yoachimik, *flowtrackd: DDoS Protection with Unidirectional TCP Flow Tracking*.
[2] Facebook Incubator, *Katran*.
[3] Cilium Authors, *Cilium*.
[4] Tu *et al.*, 'Revisiting the Open vSwitch Dataplane Ten Years Later'.

Who's using it?

Cloudflare DDoS attack scrubbing *(flowtrackd[1])*.

Meta Fast in-kernel L4-aware load balancing *(Katran[2])*.

[1] Yoachimik, *flowtrackd: DDoS Protection with Unidirectional TCP Flow Tracking*.
[2] Facebook Incubator, *Katran*.
[3] Cilium Authors, *Cilium*.
[4] Tu *et al.*, 'Revisiting the Open vSwitch Dataplane Ten Years Later'.

Who's using it?

Cloudflare DDoS attack scrubbing *(flowtrackd[1])*.

Meta Fast in-kernel L4-aware load balancing *(Katran[2])*.

Google, AWS, ... Kubernetes load balancing & security *(Cilium[3])*.

---

[1] Yoachimik, *flowtrackd: DDoS Protection with Unidirectional TCP Flow Tracking*.
[2] Facebook Incubator, *Katran*.
[3] Cilium Authors, *Cilium*.
[4] Tu *et al.*, 'Revisiting the Open vSwitch Dataplane Ten Years Later'.

Who's using it?

Cloudflare DDoS attack scrubbing *(flowtrackd[1])*.

Meta Fast in-kernel L4-aware load balancing *(Katran[2])*.

Google, AWS, ... Kubernetes load balancing & security *(Cilium[3])*.

Open vSwitch Software routing[4].

---

[1] Yoachimik, *flowtrackd: DDoS Protection with Unidirectional TCP Flow Tracking*.
[2] Facebook Incubator, *Katran*.
[3] Cilium Authors, *Cilium*.
[4] Tu *et al.*, 'Revisiting the Open vSwitch Dataplane Ten Years Later'.

Who's using it?

Cloudflare DDoS attack scrubbing *(flowtrackd[1])*.

Meta Fast in-kernel L4-aware load balancing *(Katran[2])*.

Google, AWS, ... Kubernetes load balancing & security *(Cilium[3])*.

Open vSwitch Software routing[4].

...and kernel/userland debugging via *bpftrace* (à la Dtrace).

---

[1] Yoachimik, *flowtrackd: DDoS Protection with Unidirectional TCP Flow Tracking*.
[2] Facebook Incubator, *Katran*.
[3] Cilium Authors, *Cilium*.
[4] Tu *et al.*, 'Revisiting the Open vSwitch Dataplane Ten Years Later'.

History & Details

A Little Bit of History

eBPF was once BPF – the Berkeley/BSD Packet Filter[5].

- **2-register**, 32 bit VM.

[5] McCanne and Jacobson, 'The BSD Packet Filter: A New Architecture for User-level Packet Capture'.

A Little Bit of History

eBPF was once BPF – the Berkeley/BSD Packet Filter[5].

- **2-register**, 32 bit VM.
- Early filtering for `tcpdump` etc.

[5] McCanne and Jacobson, 'The BSD Packet Filter: A New Architecture for User-level Packet Capture'.

A Little Bit of History

eBPF was once BPF – the Berkeley/BSD Packet Filter[5].

- **2-register**, 32 bit VM.
- Early filtering for `tcpdump` etc.
- Circa 1993.

[5] McCanne and Jacobson, 'The BSD Packet Filter: A New Architecture for User-level Packet Capture'.

Technical details (I)

- 64 bit ISA.

Technical details (I)

- 64 bit ISA.
- 10 registers,

Technical details (I)

- 64 bit ISA.
- 10 registers,
- Still RISC at heart – a *very* bare-bones set of instructions.

Technical details (II)

class	value	description	reference
BPF_LD	0x00	non-standard load operations	Load and store instructions
BPF_LDX	0x01	load into register operations	Load and store instructions
BPF_ST	0x02	store from immediate operations	Load and store instructions
BPF_STX	0x03	store from register operations	Load and store instructions
BPF_ALU	0x04	32-bit arithmetic operations	Arithmetic and jump instructions
BPF_JMP	0x05	64-bit jump operations	Arithmetic and jump instructions
BPF_JMP32	0x06	32-bit jump operations	Arithmetic and jump instructions
BPF_ALU64	0x07	64-bit arithmetic operations	Arithmetic and jump instructions

where $ALU = \{+, -, \times, \div, \textit{shifts \& bitwise}, \cdots\}$, with atomic modifiers.

Technical details (III)

How does most of the magic happen?

BPF Helpers.

- Entry points and types specified by hook location

```
long bpf_trace_printk(const char *fmt,
    u32 fmt_size, ...);

long bpf_skb_vlan_push(struct sk_buff *skb,
    __be16 vlan_proto,
    u16 vlan_tci);

long bpf_xdp_adjust_head(struct xdp_buff *xdp_md,
    int delta);

u32 bpf_get_prandom_u32(void);

u64 bpf_perf_event_read(struct bpf_map *map,
    u64 flags);

u64 bpf_jiffies64(void);

long bpf_tail_call(void *ctx,
    struct bpf_map *prog_array_map,
    u32 index);

// ...
```

Technical details (III)

How does most of the magic happen?
BPF Helpers.

- Entry points and types specified by hook location
- This *also* controls what kernel functions can be called – an enforced API.

```
long bpf_trace_printk(const char *fmt,
    u32 fmt_size, ...);

long bpf_skb_vlan_push(struct sk_buff *skb,
    __be16 vlan_proto,
    u16 vlan_tci);

long bpf_xdp_adjust_head(struct xdp_buff *xdp_md,
    int delta);

u32 bpf_get_prandom_u32(void);

u64 bpf_perf_event_read(struct bpf_map *map,
    u64 flags);

u64 bpf_jiffies64(void);

long bpf_tail_call(void *ctx,
    struct bpf_map *prog_array_map,
    u32 index);

// ...
```

Technical details (III)

How does most of the magic happen?
BPF Helpers.

- Entry points and types specified by hook location
- This *also* controls what kernel functions can be called – an enforced API.
- E.g., RNG, map accesses, timer & thread information.

```
long bpf_trace_printk(const char *fmt,
    u32 fmt_size, ...);

long bpf_skb_vlan_push(struct sk_buff *skb,
    __be16 vlan_proto,
    u16 vlan_tci);

long bpf_xdp_adjust_head(struct xdp_buff *xdp_md,
    int delta);

u32 bpf_get_prandom_u32(void);

u64 bpf_perf_event_read(struct bpf_map *map,
    u64 flags);

u64 bpf_jiffies64(void);

long bpf_tail_call(void *ctx,
    struct bpf_map *prog_array_map,
    u32 index);

// ...
```

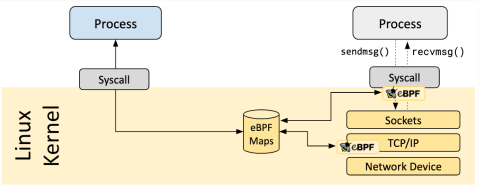
Technical details (III)

How does most of the magic happen?
BPF Helpers.

- Entry points and types specified by hook location
- This *also* controls what kernel functions can be called – an enforced API.
- E.g., RNG, map accesses, timer & thread information.
- Portable between kernel versions due to CO-RE (BTF).

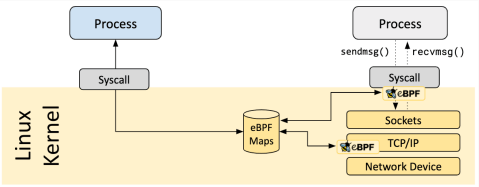
```
long bpf_trace_printk(const char *fmt,
    u32 fmt_size, ...);

long bpf_skb_vlan_push(struct sk_buff *skb,
    __be16 vlan_proto,
    u16 vlan_tci);

long bpf_xdp_adjust_head(struct xdp_buff *xdp_md,
    int delta);

u32 bpf_get_prandom_u32(void);

u64 bpf_perf_event_read(struct bpf_map *map,
    u64 flags);

u64 bpf_jiffies64(void);

long bpf_tail_call(void *ctx,
    struct bpf_map *prog_array_map,
    u32 index);

// ...
```

Technical details (IV)



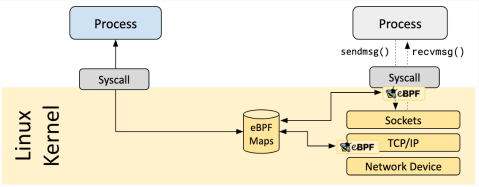
- eBPF ↔ Userland comms. via eBPF Maps.

Technical details (IV)



- eBPF ↔ Userland comms. via eBPF Maps.
- Hash tables, arrays, per-CPU maps, socket descriptor maps, program maps.

Technical details (IV)



- eBPF ↔ Userland comms. via eBPF Maps.
- Hash tables, arrays, per-CPU maps, socket descriptor maps, program maps.
- Also eBPF ↔ eBPF.

Verification

Before loading, *all programs must be verified by the kernel*:

- Bounds-checked pointer accesses.

Verification

Before loading, *all programs must be verified by the kernel*:

- Bounds-checked pointer accesses.
- Type-checked pointer accesses.

Verification

Before loading, *all programs must be verified by the kernel*:

- Bounds-checked pointer accesses.
- Type-checked pointer accesses.
- Program size limited, no unbounded loops.

Verification

Before loading, *all programs must be verified by the kernel*:

- Bounds-checked pointer accesses.
- Type-checked pointer accesses.
- Program size limited, no unbounded loops.
- Write-protection, constant-blinding of JITed code.

How do we compile to eBPF?

How do we write & interact with eBPF programs?

How do we compile to eBPF?

How do we write & interact with eBPF programs?

BCC Write in C, feed to LLVM wrapper built in Python.

Rust *redbpf*, *libbpf-cargo*, *aya*, ...

- Iffy CO-RE, Linux v6 support for redbpf.

GCC Support for C since 2020.

Cilium Write in C, launch and communicate using maps in Go.

How do we compile to eBPF? 🦀🦀

🦀🦀 How do we write & interact with eBPF programs? 🦀🦀

BCC Write in C, feed to LLVM wrapper built in Python.

Rust 🦀🦀 *redbpf*, *libbpf-cargo*, *aya*, ... 🦀🦀

- Iffy CO-RE, Linux v6 support for redbpf.

GCC Support for C since 2020.

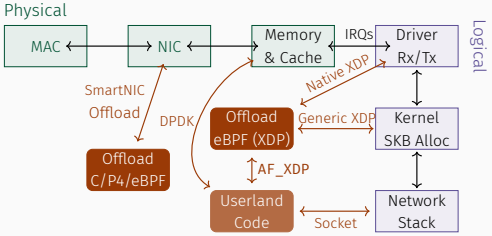
Cilium Write in C, launch and communicate using maps in Go.

...with no bias from me! 🦀

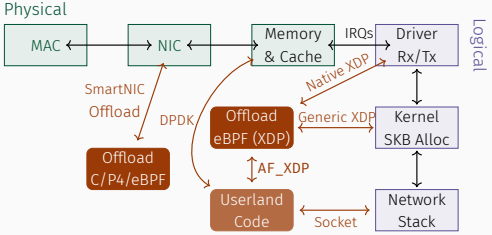
Networking

How does this relate to networking? XDP & **AF_XDP**!

- XDP is an eBPF hook attached to packet ingress

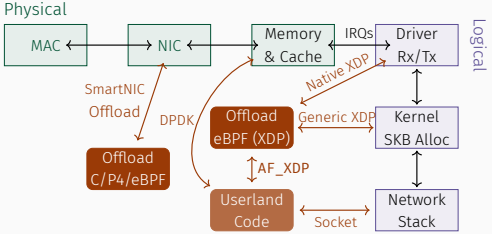


How does this relate to networking? XDP & **AF_XDP**!



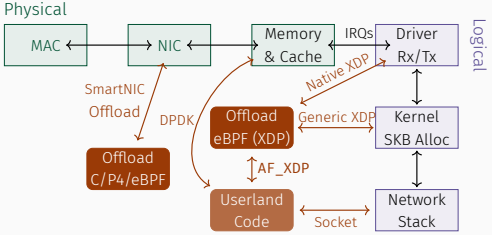
- XDP is an eBPF hook attached to packet ingress
- Not just inspect – modify.

How does this relate to networking? XDP & **AF_XDP**!



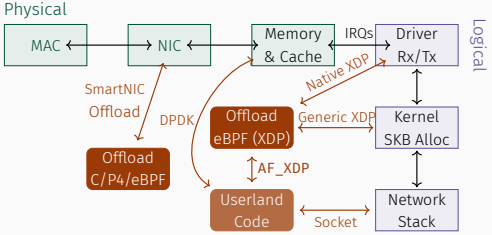
- XDP is an eBPF hook attached to packet ingress
- Not just inspect – modify.
- Variations on hook $\in$ {Offload, Driver, Generic}

How does this relate to networking? XDP & **AF_XDP**!



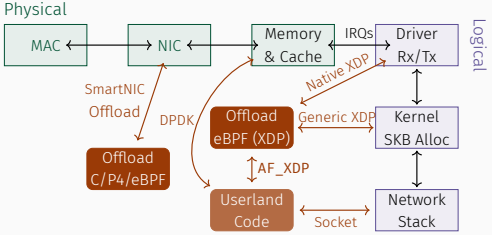
- XDP is an eBPF hook attached to packet ingress
- Not just inspect – modify.
- Variations on hook $\in$ {Offload, Driver, Generic}
  - Perf degrades gracefully according to driver support

How does this relate to networking? XDP & **AF_XDP**!



- XDP is an eBPF hook attached to packet ingress
- Not just inspect – **modify**.
- Variations on hook $\in$ {Offload, Driver, Generic}
  - Perf degrades gracefully according to driver support
- Hook can locally handle packets before forwarding to Linux stack, sending straight to (another) NIC, or drop.

How does this relate to networking? XDP & **AF_XDP**!



- XDP is an eBPF hook attached to packet ingress
- Not just inspect – **modify**.
- Variations on hook $\in$ {Offload, Driver, Generic}
  - Perf degrades gracefully according to driver support
- Hook can locally handle packets before forwarding to Linux stack, sending straight to (another) NIC, or drop.
- Since 2019: **AF_XDP** stack bypass!

Limitations

In XDP, Parallel threads limited to num of Rx queues on NIC.

Limitations

In XDP, Parallel threads limited to num of Rx queues on NIC.

Static verification means different model from e.g. Rust.

Limitations

In XDP, Parallel threads limited to num of Rx queues on NIC.

Static verification means different model from e.g. Rust.

❌

```rust
pub fn handle_pkt(pkt: &mut [u8]) -> Action {
    if pkt.len() >= 12 {
        let mut src_mac = &mut pkt[6..12];
        src_mac.copy_from_slice(&[
            0xaa, 0xbb, 0xcc,
            0xdd, 0xee, 0xff
        ]);
        // FAILS VERIFICATION
    }
    Action::Pass
}
```

✅

```rust
pub fn handle_pkt(pkt: impl Packet) -> Action {
    if let Some(src_mac) = pkt.slice_from(6, 6) {
        // bytes: &mut [u8]
        src_mac.copy_from_slice(&[
            0xaa, 0xbb, 0xcc,
            0xdd, 0xee, 0xff
        ]);
        // Passes verification!
        // Why? Trait checking pointer
        // against 'end-of-packet' ptr.
    }
    Action::Pass
}
```

Why choose this over DPDK?

- More CPU- and power-efficient than DPDK[6].

[6] Høiland-Jørgensen *et al.*, 'The eXpress data path: fast programmable packet processing in the operating system kernel'.

Why choose this over DPDK?

- More CPU- and power-efficient than DPDK[6].
- Arguably easier to write and use.

[6] Høiland-Jørgensen *et al.*, 'The eXpress data path: fast programmable packet processing in the operating system kernel'.

Why choose this over DPDK?

- More CPU- and power-efficient than DPDK[6].
- Arguably easier to write and use.
- Works on any modern Linux box.

[6] Høiland-Jørgensen *et al.*, 'The eXpress data path: fast programmable packet processing in the operating system kernel'.

Why choose this over DPDK?

- More CPU- and power-efficient than DPDK[6].
- Arguably easier to write and use.
- Works on any modern Linux box.
  - Even RPi if you recompile the kernel!

[6] Høiland-Jørgensen *et al.*, 'The eXpress data path: fast programmable packet processing in the operating system kernel'.

Why choose this over DPDK?

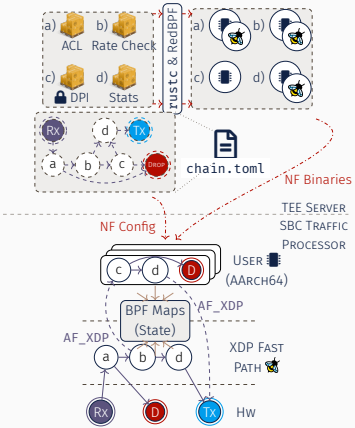
- More CPU- and power-efficient than DPDK[6].
- Arguably easier to write and use.
- Works on any modern Linux box.
  - Even RPi if you recompile the kernel!
- Performance still strong – $\mathcal{O}(20\,\mu s)$ min latency.

[6] Høiland-Jørgensen *et al.*, 'The eXpress data path: fast programmable packet processing in the operating system kernel'.

Composition

A huge limitation of eBPF programs is size. *But we have tail-calls.*

- Packet function chains in datacentres[a], with dynamic PGO[b].

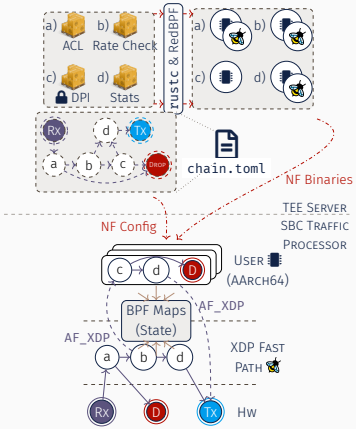


[a] Miano, Risso *et al.*, 'A Framework for eBPF-Based Network Functions in an Era of Microservices'.
[b] Miano, Sanaee *et al.*, 'Domain specific run time optimization for software data planes'.

Composition

A huge limitation of eBPF programs is size. *But we have tail-calls.*

- Packet function chains in datacentres[a], with dynamic PGO[b].
- Doable with more constraints on weaker machines – lat-tput tradeoffs (right).



[a] Miano, Risso *et al.*, 'A Framework for eBPF-Based Network Functions in an Era of Microservices'.
[b] Miano, Sanaee *et al.*, 'Domain specific run time optimization for software data planes'.

Takeaways:

eBPF is a powerful tool for accelerating networked services and host instrumentation.
Easy to program from your favourite systems programming languages!
Portable and actively developed.
A hot topic! *Active SIGCOMM CFP* for networks.

Questions?

ARISTA
ks@arista.com
ksArista
https://mcfelix.me

References i

- Cilium Authors. ***Cilium. Linux Native, API-Aware Networking and Security for Containers.*** 2022. URL: `https://cilium.io` (visited on 29/12/2022).
- Facebook Incubator. ***Katran. A high performance layer 4 load balancer.*** 2020. URL: `https://github.com/facebookincubator/katran` (visited on 12/04/2023).

References ii

Høiland-Jørgensen, Toke, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern and David Miller. **'The eXpress data path: fast programmable packet processing in the operating system kernel'.** In: *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2018, Heraklion, Greece, December 04-07, 2018*. Ed. by Xenofontas A. Dimitropoulos, Alberto Dainotti, Laurent Vanbever and Theophilus Benson. ACM, 2018, pp. 54–66. DOI: 10.1145/3281411.3281443. URL: https://doi.org/10.1145/3281411.3281443.

References iii

- McCanne, Steven and Van Jacobson. **'The BSD Packet Filter: A New Architecture for User-level Packet Capture'.** In: *Proceedings of the Usenix Winter 1993 Technical Conference, San Diego, California, USA, January 1993*. USENIX Association, 1993, pp. 259–270. URL: https://www.usenix.org/conference/usenix-winter-1993-conference/bsd-packet-filter-new-architecture-user-level-packet.
- Miano, Sebastiano, Fulvio Risso, Mauricio Vásquez Bernal, Matteo Bertrone and Yunsong Lu. **'A Framework for eBPF-Based Network Functions in an Era of Microservices'.** In: *IEEE Trans. Netw. Serv. Manag.* 18.1 (2021), pp. 133–151. DOI: 10.1109/TNSM.2021.3055676. URL: https://doi.org/10.1109/TNSM.2021.3055676.


Miano, Sebastiano, Alireza Sanaee, Fulvio Risso, Gábor Rétvári and Gianni Antichi. **'Domain specific run time optimization for software data planes'.** In: *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*. Ed. by Babak Falsafi, Michael Ferdman, Shan Lu and Thomas F. Wenisch. ACM, 2022, pp. 1148–1164. DOI: 10.1145/3503222.3507769. URL: https://doi.org/10.1145/3503222.3507769.

References v

- Tu, William, Yi-Hung Wei, Gianni Antichi and Ben Pfaff. **'Revisiting the Open vSwitch Dataplane Ten Years Later'.** In: *ACM SIGCOMM 2021 Conference, Virtual Event, USA, August 23-27, 2021*. Ed. by Fernando A. Kuipers and Matthew C. Caesar. ACM, 2021, pp. 245–257. DOI: 10.1145/3452296.3472914. URL: https://doi.org/10.1145/3452296.3472914.
- Yoachimik, Omar. ***flowtrackd: DDoS Protection with Unidirectional TCP Flow Tracking.*** 14th July 2020. URL: https://blog.cloudflare.com/announcing-flowtrackd/ (visited on 29/12/2022).