Galette: a Lightweight XDP Dataplane on your Raspberry Pi

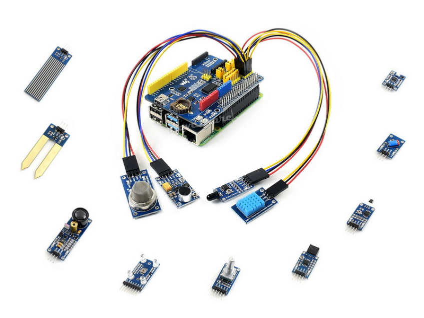
Kyle A. Simpson, Chris Williamson, Douglas J. Paul, Dimitrios P. Pezaros
kylesimpson1@acm.org
FelixMcFelix https://mcfelix.me

13th June, 2023

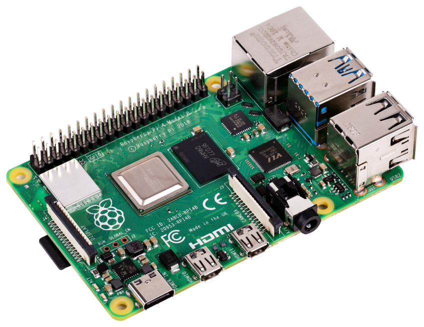
University of Glasgow

Securing Sensor & IoT Gateway Networks

- Security – ingress/egress packet processing by *network functions*.
  - IP layer – Firewalls, DPI, ACLs...
  - Middleboxes a bad fit.
  - Needs to be reconfigurable – attacks and security context evolve.
- Ideally in-situ.
  - Dynamic/retrofitted.
  - But limited space + power in the field.
  - Physically vulnerable!
- *Sensor networks have low data rates!*



Fast, cheap, and secure IoT Defence – pick 3?



- Single-board compute like RPis are small, capable, affordable! Cheap!
  - See also: NUCs (££), Jetsons (£££).
  - *Linux-based*: Easy(/ier) to target and write for. We also get kernel network stack advancements.
  - **Different CPU architectures.**
- Project goals:
  - Fast! Low-latency, quickly reconfigurable.
  - Secure! efficient NFV code gen from *memory-safe languages*.

Galette's Research Objectives

Galette puts **effective** eBPF packet processing into **edge computers**.

1. What *specialisations* does XDP Function Chaining need to *best suit SBCs*?
    - Split userland-XDP pipeline.
2. How do we make eBPF + native compile from memory-safe systems languages easy? And portable across 'native'?
3. How efficient is it on RPi/NUC?

Galette's Research Objectives

Galette puts **effective** eBPF packet processing into **edge computers**.

1. What *specialisations* does XDP Function Chaining need to *best suit SBCs*?
    - Split userland-XDP pipeline.
    - Many userland pipes!
2. How do we make eBPF + native compile from memory-safe systems languages easy? And portable across 'native'?
3. How efficient is it on RPi/NUC?

GALETTE's Research Objectives

GALETTE puts **effective** eBPF packet processing into **edge computers**.

1. What *specialisations* does XDP Function Chaining need to *best suit SBCs*?
    - Split userland-XDP pipeline.
    - Many userland pipes!
2. How do we make eBPF + native compile from memory-safe systems languages easy? And portable across 'native'?
    - One Rust program per NF $\implies$ eBPF + native.
3. How efficient is it on RPi/NUC?

GALETTE's Research Objectives

GALETTE puts **effective** eBPF packet processing into **edge computers**.

1. What *specialisations* does XDP Function Chaining need to *best suit SBCs*?
    - Split userland-XDP pipeline.
    - Many userland pipes!
2. How do we make eBPF + native compile from memory-safe systems languages easy? And portable across 'native'?
    - One Rust program per NF $\implies$ eBPF + native.
    - Easier, unified API.
3. How efficient is it on RPi/NUC?

Galette's Research Objectives

Galette puts **effective** eBPF packet processing into **edge computers**.

1. What *specialisations* does XDP Function Chaining need to *best suit SBCs*?
    - Split userland-XDP pipeline.
    - Many userland pipes!
2. How do we make eBPF + native compile from memory-safe systems languages easy? And portable across 'native'?
    - One Rust program per NF $\implies$ eBPF + native.
    - Easier, unified API.
    - Simple, dynamic chain format.
3. How efficient is it on RPi/NUC?

GALETTE's Research Objectives

GALETTE puts **effective** eBPF packet processing into **edge computers**.

1. What *specialisations* does XDP Function Chaining need to *best suit SBCs*?
   - Split userland-XDP pipeline.
   - Many userland pipes!
2. How do we make eBPF + native compile from memory-safe systems languages easy? And portable across 'native'?
   - One Rust program per NF $\implies$ eBPF + native.
   - Easier, unified API.
   - Simple, dynamic chain format.
3. How efficient is it on RPi/NUC?
   - Better latency, throughout, power use than AF_PACKET...

GALETTE's Research Objectives

GALETTE puts **effective** eBPF packet processing into **edge computers**.

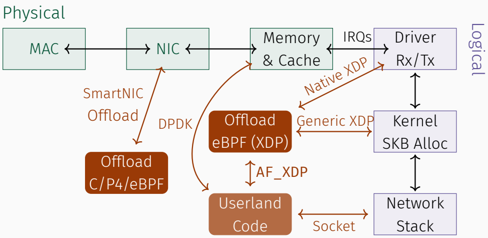
1. What *specialisations* does XDP Function Chaining need to *best suit SBCs*?
   - Split userland-XDP pipeline.
   - Many userland pipes!
2. How do we make eBPF + native compile from memory-safe systems languages easy? And portable across 'native'?
   - One Rust program per NF $\implies$ eBPF + native.
   - Easier, unified API.
   - Simple, dynamic chain format.
3. How efficient is it on RPi/NUC?
   - Better latency, throughout, power use than AF_PACKET...
   - **...without polling.**

Background

eBPF: What and Why?

- Simple register machine VM (user-written) code, derived from BPF.
- Modern use – Kernel hooks, perf instrumentation, debugging
- JIT compiled
- Kernel-verified
  - Bounds-checked pointer accesses
  - Program size limited, no unbounded loops
  - Syscalls (*eBPF helpers*) exposed based on hook point

Network stack improvements: XDP



- eBPF hook attached to packet ingress
- Variations on hook $\in$ {Offload, Driver, Generic}
  - Perf degrades gracefully according to driver support
- Hook can modify & inspect packets before forwarding to Linux stack, sending straight to (another) NIC, or drop.
- Since 2019: **AF_XDP** stack bypass!

Q1: Specialising AF_XDP Function Chaining for SBCs

The Unique Challenges of SBCs

- **Problem:** ‘Best’ low latency processing (DPDK) is expensive – CPU, power, *HW support*.
- **Problem:** Mismatch of HW queues to physical cores:
  - Soln: load balance and place high-latency NFs in userland.
- **Problem:** XDP hooks only on ingress (*for now*):

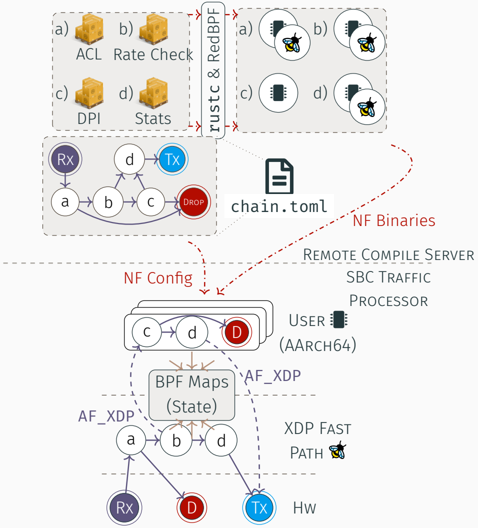
The Unique Challenges of SBCs

- **Problem:** ‘Best’ low latency processing (DPDK) is expensive – CPU, power, *HW support*.
- **Problem:** Mismatch of HW queues to physical cores:
  - Soln: load balance and place high-latency NFs in userland.
  - …also, don’t pass packets back to kernel-space.
- **Problem:** XDP hooks only on ingress (*for now*):

The Unique Challenges of SBCs

- **Problem:** 'Best' low latency processing (DPDK) is expensive – CPU, power, *HW support*.
- **Problem:** Mismatch of HW queues to physical cores:
    - Soln: load balance and place high-latency NFs in userland.
    - ...also, don't pass packets back to kernel-space.
- **Problem:** XDP hooks only on ingress (*for now*):
    - Soln: Write an individual NF *once*, compile for both envs, and replicate NFs as needed.

GALETTE Design: Bird's eye view



- Two-tier approach—XDP & User.
- Composable NFs – graph structure.
- Critical or high performance NFs go into XDP:
  - Low latency for most packets.
  - Chain with XDP tail calls.
- Rare 'slow-path' still kernel bypass:
  - Expensive & proprietary code.
  - Only for candidate attack traffic.
- Reconfigurable, dynamic.
- Remote-compiled.

How does this differ from other frameworks?

In Security? SafeBricks[1], AuditBox[2] or similar.

- ...No SGX support in devices of interest.

In eBPF/XDP space? Polycube[3]!

- Built around datacentres – we often have just one HW queue for a NIC.
- ...so we use more userland pipes to scale to the extra cores we *do* have.

---

[1] Poddar *et al.*, 'SafeBricks: Shielding Network Functions in the Cloud'.
[2] Liu *et al.*, 'Don't Yank My Chain: Auditable NF Service Chaining'.
[3] Miano *et al.*, 'A Framework for eBPF-Based Network Functions in an Era of Microservices'.

- **Problem:** Can send packet over AF_XDP, but *no context on what the next (callee) NF is.*
    - Polycube's solution inadequate: one discrete userland component per *cube*.
- Soln: Adjust headroom of packets, write in ID and action of caller.

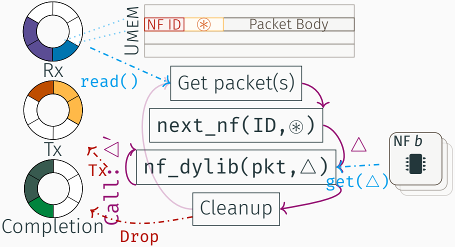


Figure 1: *Packet processing in the XDP Fast Path (NF maps omitted).*

Q2: Easy Joint-Compile (eBPF + Native) from Rust 🦀

Skeleton details

- Consistent NF API for both XDP/userland.
- Rust compiler should be able to enforce…
  - `#![forbid(unsafe_code)]` (or similar cargo tooling) on NF module crates,
  - all NF branches specified.
- All compilation on external server.
  - SBC too constrained.
  - If compile-server is TEE-equipped, can attest compiler/code etc. following SotA!

```rust
#![no_std]
pub use nf::*;

#[maps]
pub struct Maps { count: (u32, u64) }

pub enum Action { Continue }

pub fn packet<M1>(
    mut pkt: impl Packet,
    mut maps: Maps<M1>
) -> Action where M1: Map<u32, u64>,
{
    if let Some(bytes) = pkt.slice(12) {
        // bytes: &mut [u8]
        let (src_mac, rest) = bytes.split_at_mut(6);
        src_mac.swap_with_slice(&mut rest[..]);

        if let Some(n) = maps.count.get(&0) {
            maps.count.put(&0, &(n + 1));
        }
    }

    Action::Continue
}
```

`mod.rs`: A counting macswap function

A Service Funtion Chain: `security.toml`

```toml
# -- NF & Map definitions --
[functions.access-control.maps]
allow-list = {
  type = "lpm-trie",
  size = 65535
}

[functions.weak-classifier]
maps = { flow-state = "_" }

[functions.dpi]
maps = { flow-state = "_" }
disable_xdp = true

[maps.flow-state]
type = "hash_map"
size = 65535
```

```toml
# -- Chain definition --
[[links]]
from = "rx"
to = ["access-control"]

[[links]]
from = "access-control"
to = ["tx", "weak-classifier"]

[[links]]
from = "weak-classifier"
to = ["tx", "!dpi", "drop"]

[[links]]
from = "dpi"
to = ["tx", "drop"]
```

A Peek Behind The Curtain

```rust
pub struct PodData {
  pub a: u8,
  pub b: bool,
  pub c: u64,
}

#[maps]
pub struct TestMaps {
  plain: (u32, u64),
  composite: (u32, PodData),
}
```

$\Longrightarrow$

```rust
pub type NfKeyTy0 = u32;
pub type NfKeyTy1 = u32;
pub type NfValTy0 = u64;
pub type NfValTy1 = PodData;

pub struct TestMaps<NfMapField0, NfMapField1>
where
  NfMapField0: Map<u32, u64>,
  NfMapField1: Map<u32, PodData>,
{
  pub plain: NfMapField0,
  pub composite: NfMapField1,
}
```

And templating code parses any `struct`s tagged `#[maps]` to count & *generate output crates!*

Q3: Performance

Setup

Baselines

Non-Polling	Polling
GALETTE (XDP)	GALETTE (all)
GALETTE (AF_XDP)	AF_PACKET
GALETTE (Split)	DPDK (NUC)

Machines

- Raspberry Pi Model 3B (100 Mbit/s),
- Intel i7 NUCs (1 Gbit/s).

NFs

- Macswap,
- Blocking workloads ($\leq$1 ms).

Why?

- Power Draw on Pi, Latency/Throughput for all.
- Different architectures.

High-level Results

- Pure XDP & **AF_XDP** more CPU-efficient than polling baselines (line-rate on NUC).
- On RPi? Better than **AF_PACKET** on all metrics without polling.
  - Limited by fused Eth+USB controller.
- XDP-Userland split prevents packet stalls with (conditionally) heavy chains.
  - Limited by fused Eth+USB controller.

More detail? Please check out our paper!

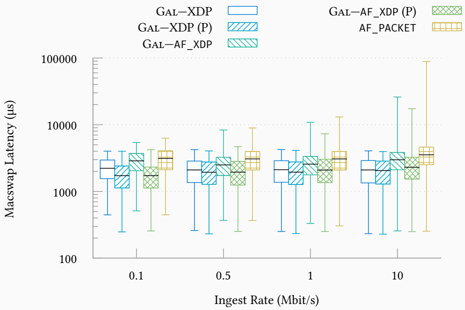


Figure 2: *RPi 64 B packet latencies.*

Takeaways:

Cheap NFs at the edge: SBCs for packet processing.

Low-latency and fast: XDP path for majority of traffic, early & cheap anomaly checks, power savings.

Secure: Rust NFs means memory safety *and* performant.

Easy to write: *native and XDP* portable NFs in Rust.

Questions?

kylesimpson1@acm.org

FelixMcFelix https://mcfelix.me

University of Glasgow

NETLAB
NETWORKED SYSTEMS RESEARCH LABORATORY
University of Glasgow | School of Computing Science