XDP Scaling Updates From the Field

One small step for your server, one giant leap for your distributed network

Ivan Koveshnikov
Gcore, 2023

Is scaling really an issue for XDP/eBPF?

- Community-recommended XDP/eBPF approach – skeleton – streamlines the delivery and management of eBPF programs. And libxdp allows to manage program.
- CO-RE addresses most version-specific kernel issues when adopting eBPF.
- XDP provides a generic interface on top of all NICs.
- XDP/eBPF performance is constrained when operating on a single server. Isn't scaling just about a deployment across multiple nodes?

So, is there still anything to talk about?

CORE

This XDP is just 100 lines of code!

A small eBPF/XDP core gives 80% of the value.

The remaining 80% of the work happens in the user space. It is rarely discussed in talks or articles, as it's thought to be generic enough for all distributed systems. Extra 80% of work is required to support the operations.

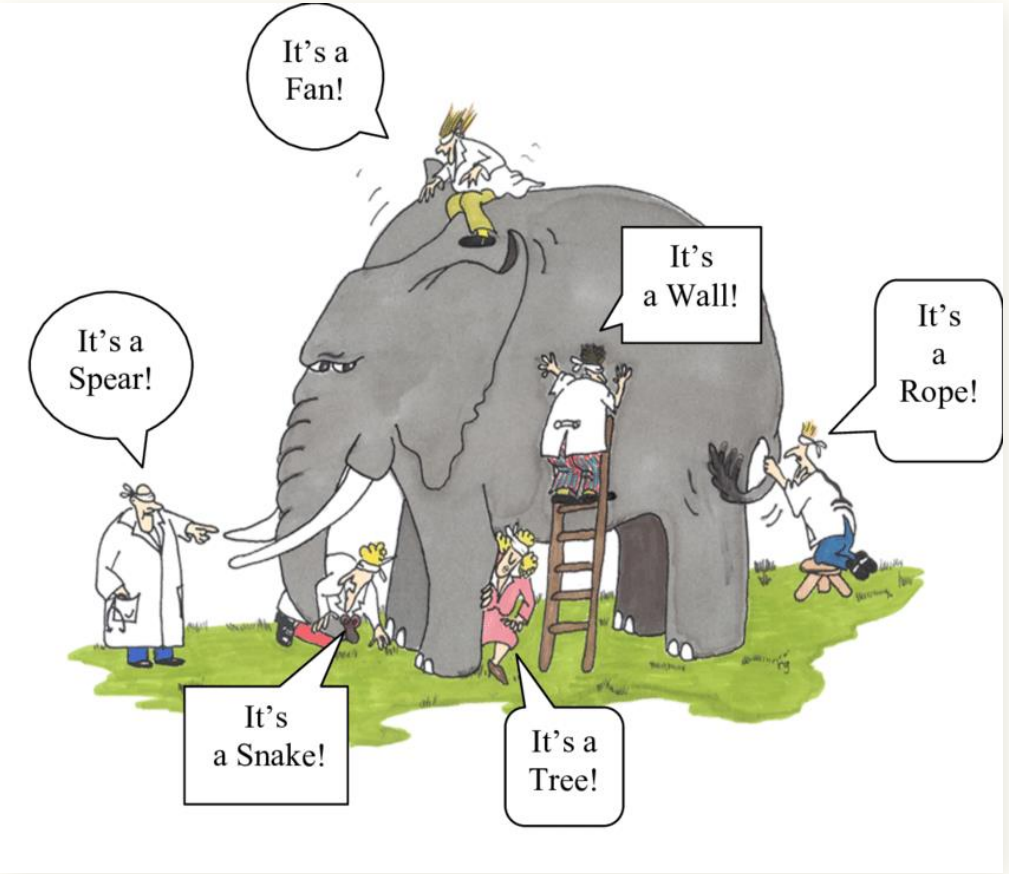
Let's delve into the main criteria for employing XDP/eBPF in networking
and identify key considerations for your design.

CORE

The life of XDP/eBPF in distributed service provider networks

The answer depends on the nature of yours eBPF application.

- Kernel compatibility: A very limited range of kernel versions and only 1-2 target operating systems
- Fast configuration updates
- Runtime code updates
- Zero downtime and no session drops
- Resource isolation
- Network visibility
- Monitoring
- CI and test automation
- Vendor lock-ins



CORE

The basis of a scalable XDP application: configuration and reloading mechanisms

Configuration:

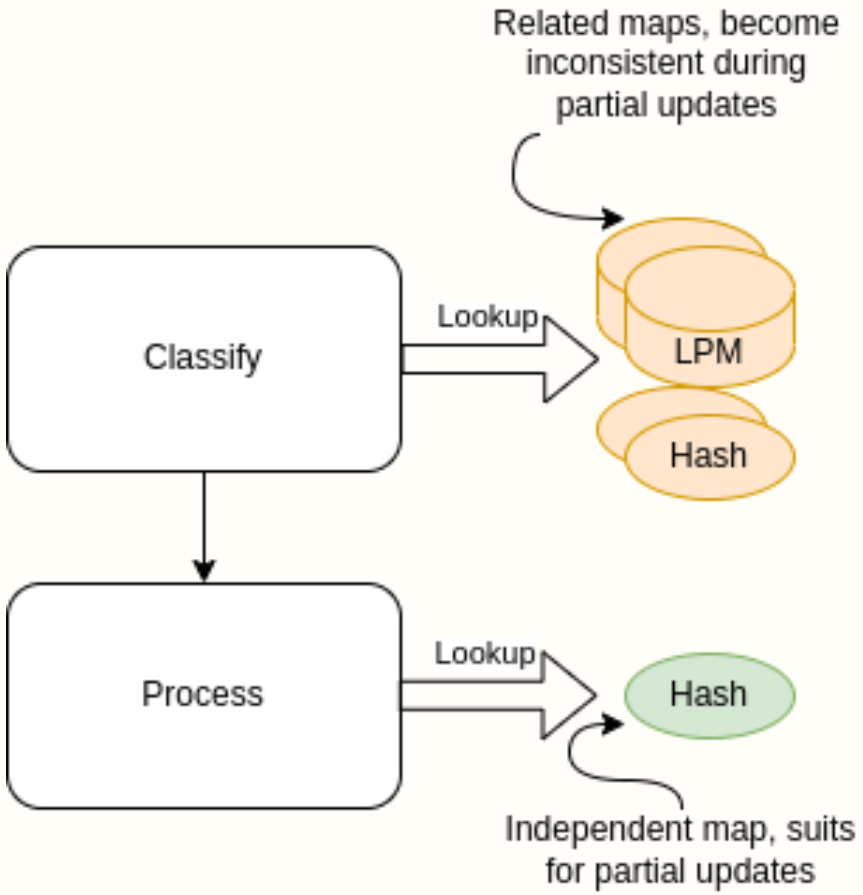
- eBPF maps are a strong asset, offering atomic modifications, lockable entries, and automatic memory management.
- Problems arise when EBPF maps start to depend on one another.
- Multiple strategies are possible to overcome these challenges.

Reloading:

- Hot code reloading: eBPF code must be updated without interrupting the service.
- Some state information must be preserved between reloads.
- Different code often means a different type system.

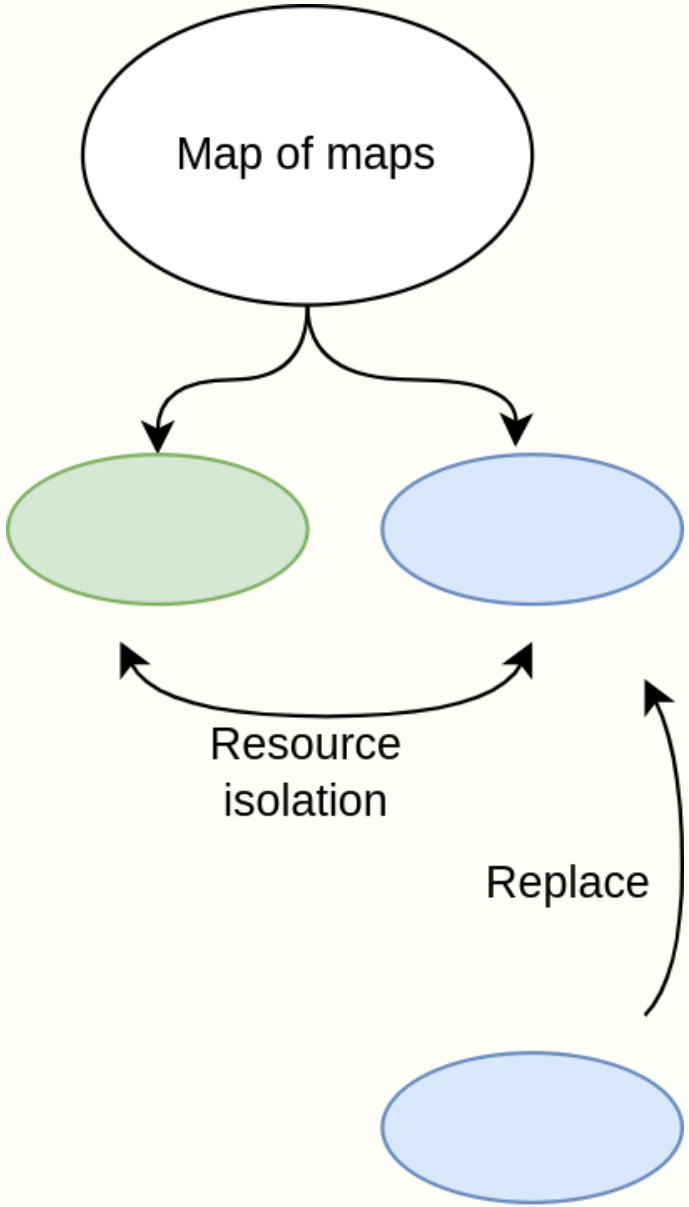
Base configuration strategy: entry-by-entry modification

- The most obvious approach, dictated by eBPF structure
- A very fast and efficient way to apply small changes
- Inefficient on major updates
- Some maps may become inconsistent until all the changes are applied
- Sharing maps between different configuration entities may cause resource concurrency
- More complicated user space code as analysis is required to transition from current to new configuration



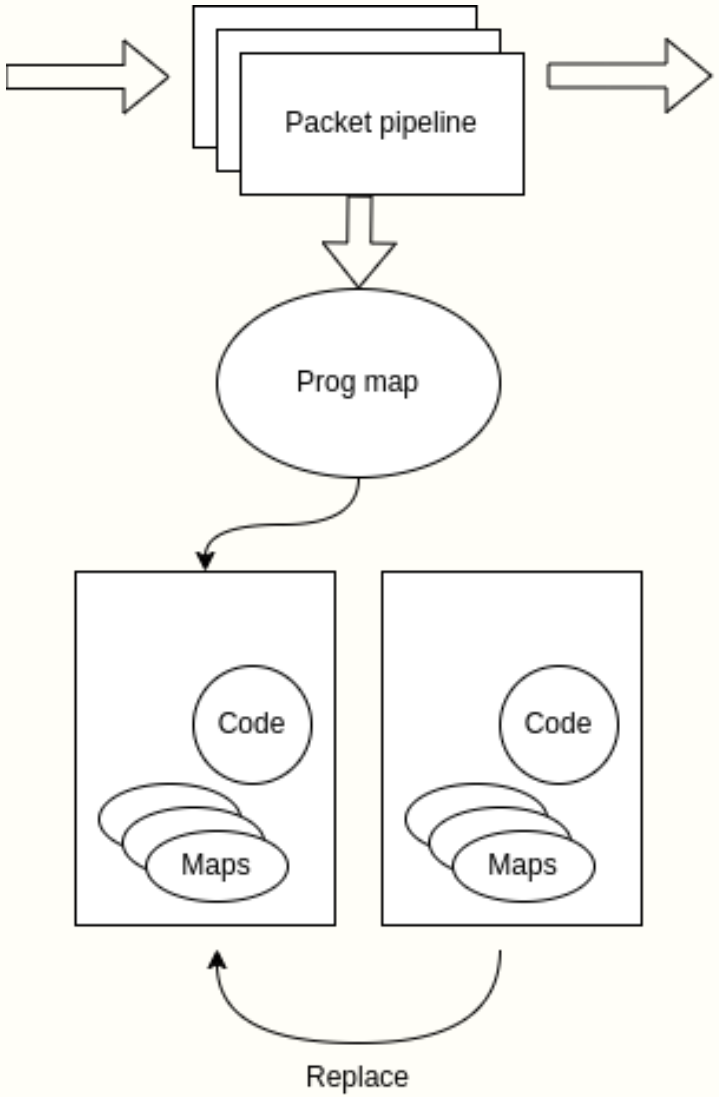
Base configuration strategy: map hot swap

- Large single shot updates: maps of maps
- A very fast and efficient method for bulky changes
- Inefficient on minor updates
- Dependencies between maps may result in system inconsistencies
- As map of maps is already in use, it can serve as a resource isolation mechanism
- Different approach to user space: no need to analyse the changes, but need to create, pin, and use new map instances



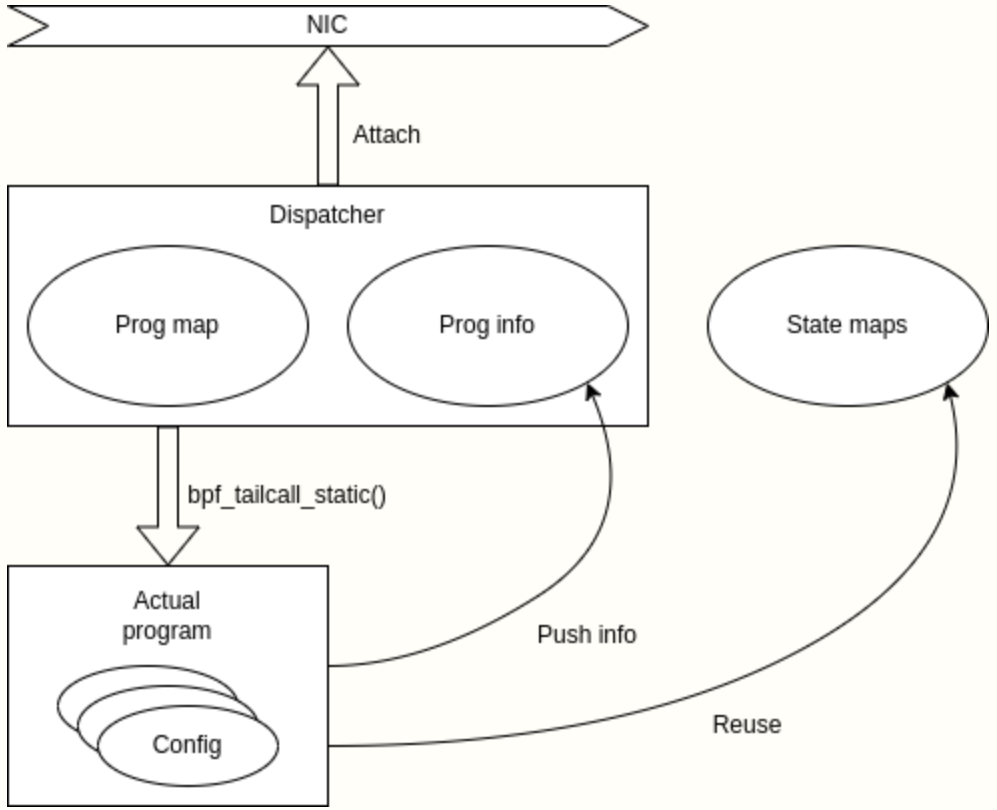
Base configuration strategy: program hot swap

- More «UNIX way» approach
- The king of bulky updates: update several maps with a single operation
- Code update as a bonus
- Some configuration can be pushed into read-only section
- Inefficient on minor updates
- Ideal for partial reloads; particularly beneficial when updating sections of complex eBPF programs, which are inherently modular
- Less manipulation on maps content, more manipulation on programs and pipeline



XDP code reload

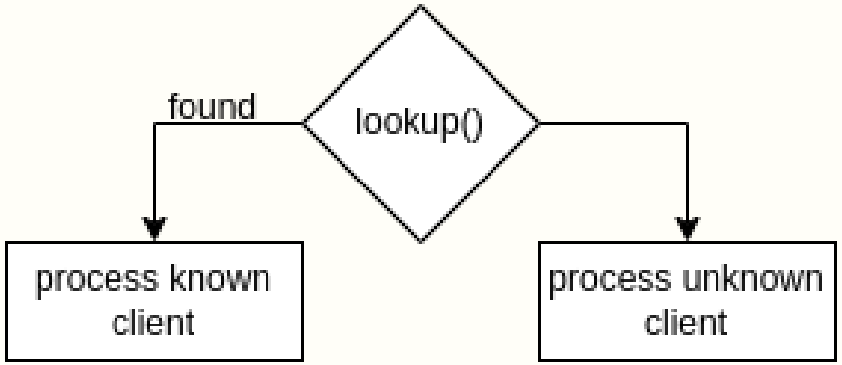
- Update code with zero downtime and no session drops
- Split configuration data and state data; reconfigure the first, reuse the latter
- BTF information may not be always useful for state maps
- To switch processing from one program to another, atomically overrwrite XDP attachments or use libxdp-like chained programs
- Free features: test new configurations, quick rollbacks, partial switches
- Use metadata to identify instances of your application
- A custom eBPF loader is essential due to the complex logic of pins, attachments, and configuration updates
- Don't forget to reload the user space code!

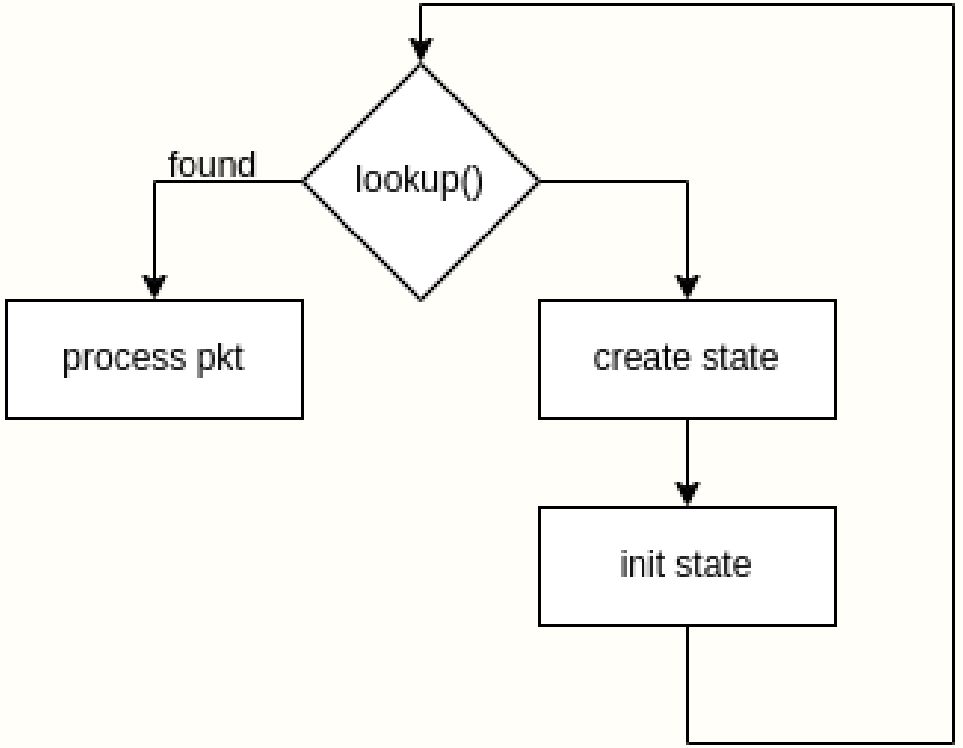


Challenges in eBPF functionality: what's missing?

bpf_map_lookup_or_update()

Frequently, a map entry is created in a hot path and requires immediate access.

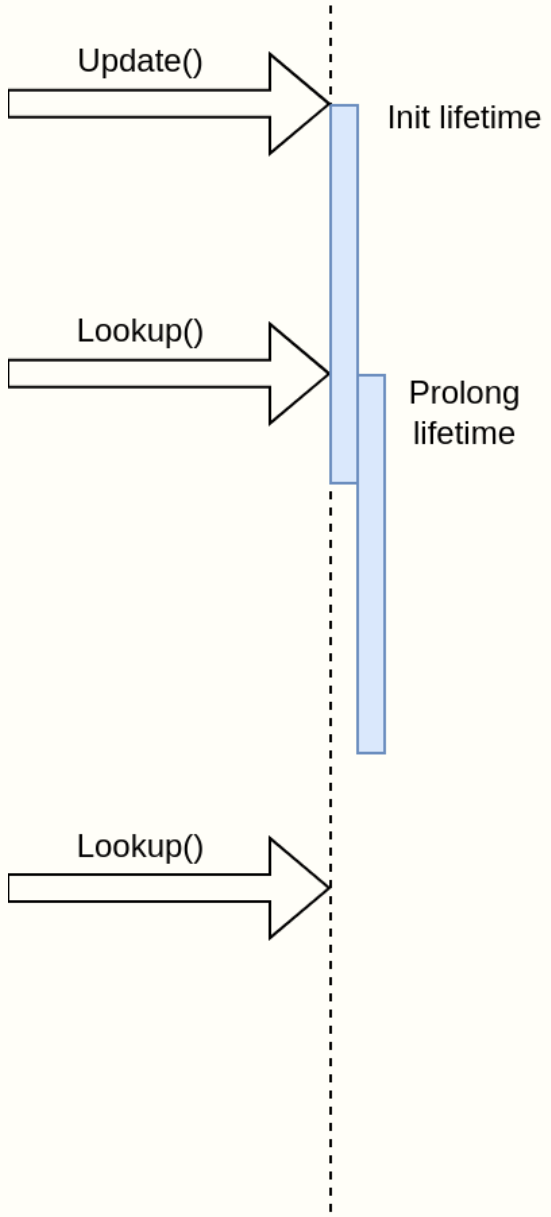




Challenges in eBPF functionality: what's missing?

Map entry lifespan for LRU: invalidate stale entries.

While BPF maps handle memory management effectively, entry lifetime may depend on application. Generic solution can be complicated. This is primarily due to the need for spinlocks and atomic operations on individual map elements.



Brief eBPF code of 100 lines?
Sure, but massive user space code!

- The more complicated the configuration model, the more sophisticated the user space.
- When using hot code reload, be aware that multiple instances of the same eBPF program could be loaded and pinned. user space code should be designed to account for this.
- There are no high-level libraries specifically designed to handle hot reloads or manage program pins and attaches. Typically, this functionality is missing, and developers have to build it with low-level libbpf.
- The user space must be able not only to configure the hot path but also read the current configuration.
- Different configuration strategies look almost the same in eBPF, but totally different in the user space.

And this is just the beginning!

Configuration delivery

- Configuration must be delivered within short and predictable amount of time.
- It may be applied unsuccessfully. How to recover from faulty state and not propagate the error?
- Versioning and backward compatibility is a must.

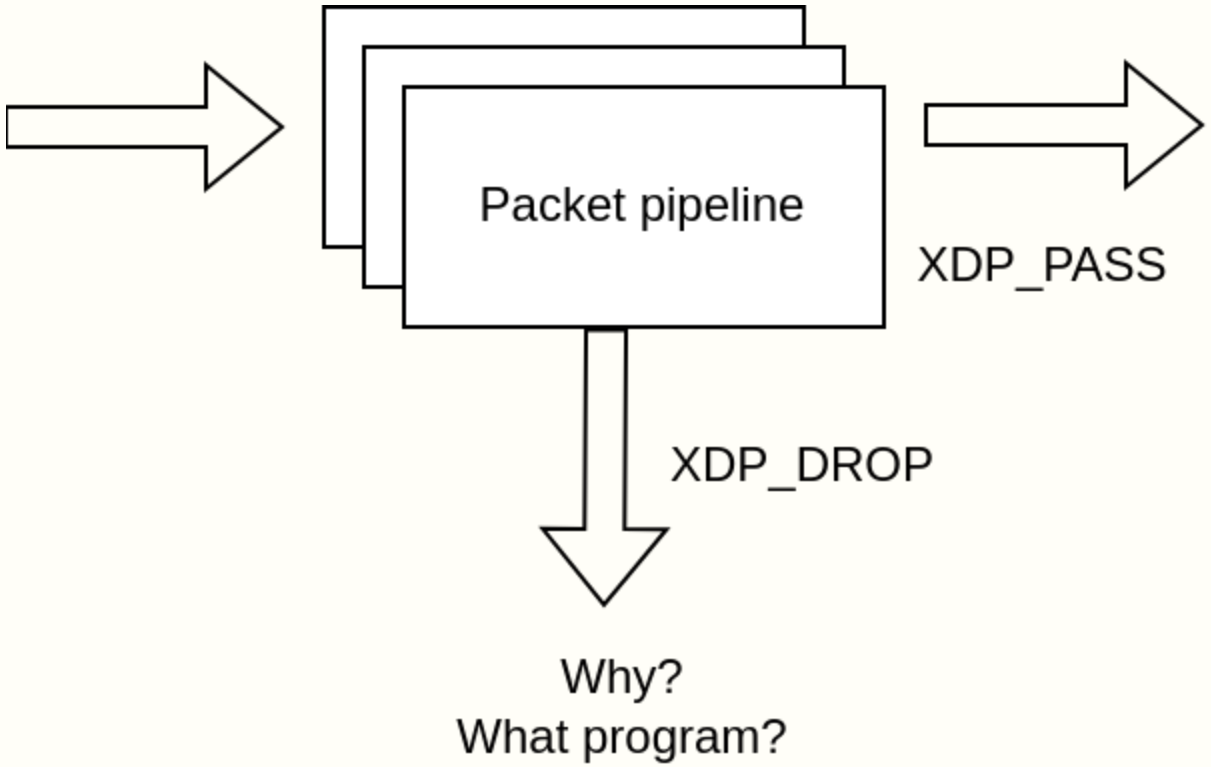
This question matters only on scale and on complex configurations with frequent changes.

CORE

Tracing in hot path

You have tuned an XDP code to maximise performance? Time to bloat the code!

- XDP_DROP and XDP_PASS are too generic. How many real verdicts your program has?
- What was the sub-program that dropped the packet in your complex pipeline?
- What is the statistics per configuration entry?
- How to debug certain traffic flows? Dumping via `Xdp_dump` is too generic.




```
Packet comments
  cus: 601, pol: 1121, mitigation: cm_cs, verdict: [DROP]
    [Expert Info (Comment/Comment): cus: 601, pol: 1121, mitigation: cm_cs,
Frame 4: 535 bytes on wire (4280 bits), 535 bytes captured (4280 bits) on int
  Section number: 1
  Interface id: 1 (sifter@fexit)
  Interface queue: 23
  Packet flags: 0x00000001
  Packet id: 2
  Verdict: , eBPF_XDP (2)
     eBPF XDP: XDP_DROP (1)
  Encapsulation type: Ethernet (1)
```

Monitoring and network observability

- Health monitoring: periodic maintenance of some maps requires ongoing usage monitoring. Is the map size adequate?
- Different entities may store resources in shared maps and will end up competing for resources
- A map entry may still be present, but its lifetime may be expired
- Insertion rate may differ from usage delta
- Map monitoring must consider the configuration
- Observability can be a valuable product in its own right, especially in the context of service providers.
- Observability may work as a feedback for traffic processing.
- What will happen with observability counters during reloads?

You can’t hide from your hardware

- Do packet drops happen due to lack of CPU time or due to pipeline implementation inside NICs hw/fw/drivers?
- Different approaches to NIC counters are well-known issues that exacerbate difficulties in monitoring and alerting. High-level monitoring suites often unify counters based on their own interpretations, causing ambiguity when different NICs are involved.
- Understanding how to configure the NIC pipeline is key. Configuration options may be available via firmware, often in an undocumented manner.
- Performance tuning and NIC monitoring may become new parts of your user space code.
- Tune NIC first and then attach or vice versa?

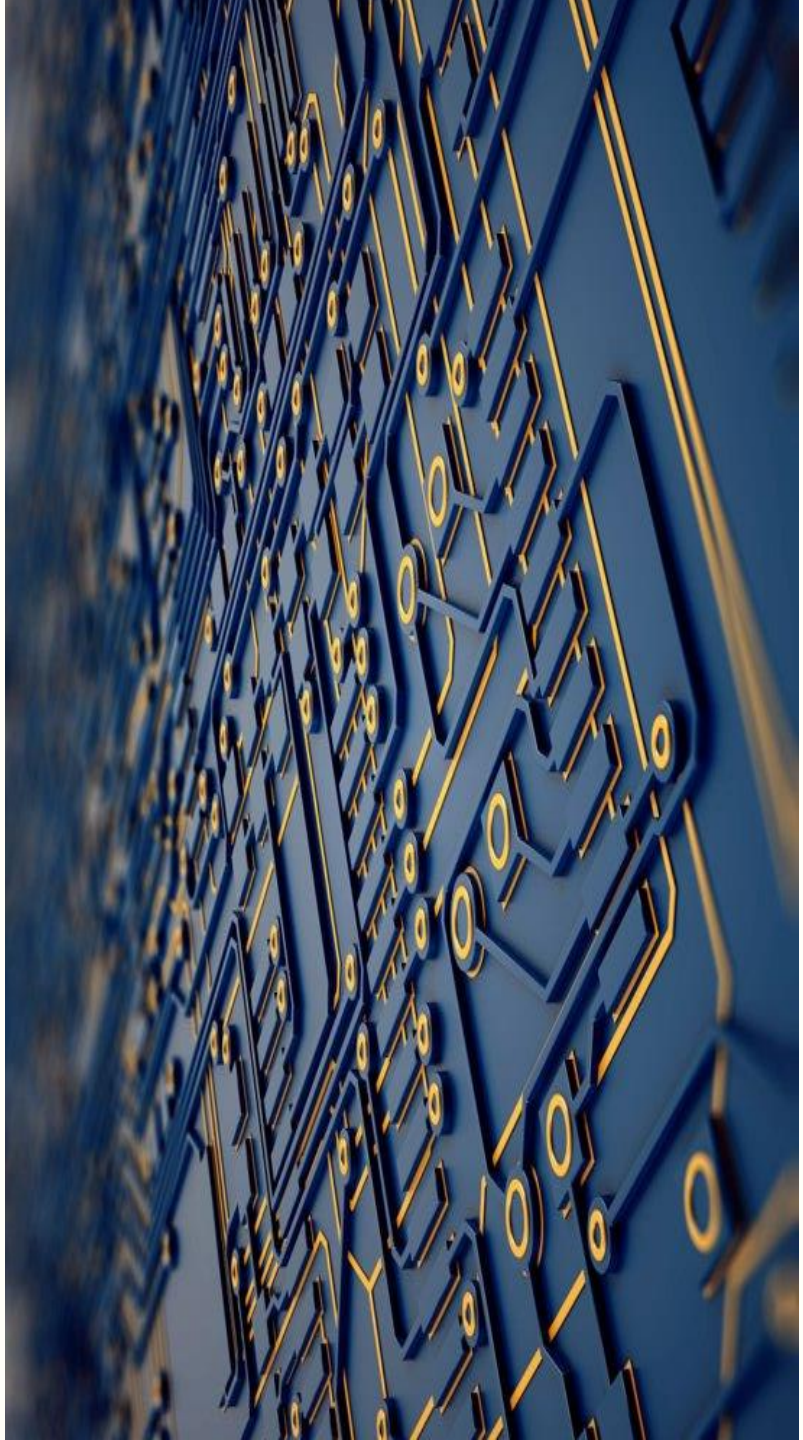
XDP is not NIC-agnostic!

A battle of load balancers

Know your network and layers of load balancing:

- Network
- NIC
- Yours XDP/eBPF
- Next application

How to trick balancers to minimize the latency overhead?



Test automation

- bpf_run_prog is super handy... for small programs.
- Big eBPF programs' behaviour may vary depending on timers, packet sequence, observability counters, and entry lifetime.
- Functional-like tests that emulate traffic flows and all the user space daemons are preferred. These tests necessitate the loading of eBPF code into the kernel for a more realistic evaluation.
- Dedicated performance tests are a must!
- CI environments may have more limitations than production servers, such as reduced memory and lower performance. Adapting eBPF code to different hardware specifications on-the-fly becomes essential.

Disabling optimisation is an optimisation

If there is worst scenario, it will happen.

- Using an LRU hash table as a cache? What will happen when it will get full?
- NO_COMMON_LRU vs no common table?
- High insertion rate into tables

Summary

- eBPF/XDP program must be reloadable and easy configurable
- Large user space code base for lifetime management
- Significant efforts for operations

CORE

CORE

Thank you!

Go global faster with Gcore CDN

PCI DSS COMPLIANT

MADE IN LUXEMBOURG

GUINNESS WORLD RECORDS

gcore.com