BPF

Turning Linux into a Microservices-aware Operating System

About the Speaker

Thomas Graf

- Linux kernel developer for ~15 years working on networking and security
- Helped write one of the biggest monoliths ever
- Worked on many Linux components over the years (IP, TCP, routing, netfilter/iptables, tc, Open vSwitch, …)
- Creator of Cilium to leverage BPF in a cloud native and microservices context
- Co-Founder & CTO of the company building Cilium

Agenda

- **Evolution of running applications**
  - From single task processes to microservices
- **Problems of the Linux kernel**
  - The kernel
- **What is BPF?**
  - Turning Linux into a modern, microservices-aware operating system
- **Cilium - BPF-based networking security for microservices**
  - What is Cilium?
  - Use Cases & Deep Dive
- **Q&A**

Evolution: Running applications

Dark Age: Single tasking

The simple age.

Multi tasking

Split the CPU and memory. Shared libraries, package management, Linux distributions.

Virtualization

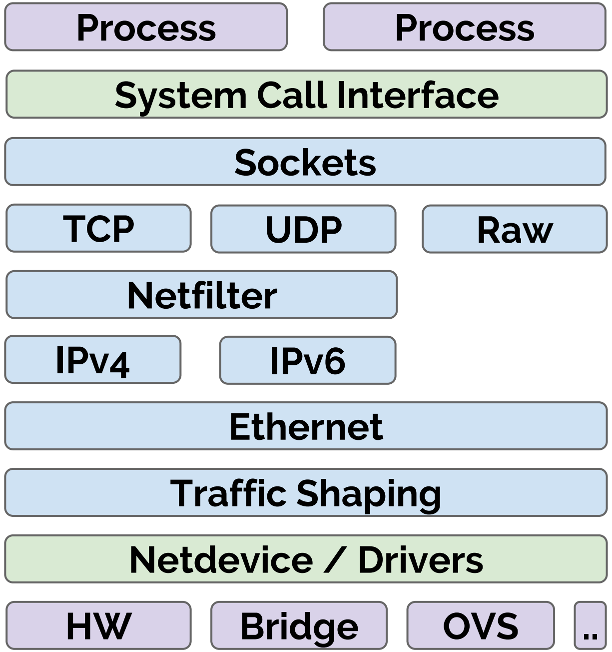
Ship the OS together with application and run it in a VM for better resource isolation. Virtualized hardware and software defined infrastructure.

Microservices Containers

Back to a shared operating system. Applications directly interact with the host operating system again.

Problems of the Linux Kernel in the age of microservices

Problem #1: Abstractions



The Linux kernel is split into layers to provide strong abstractions.

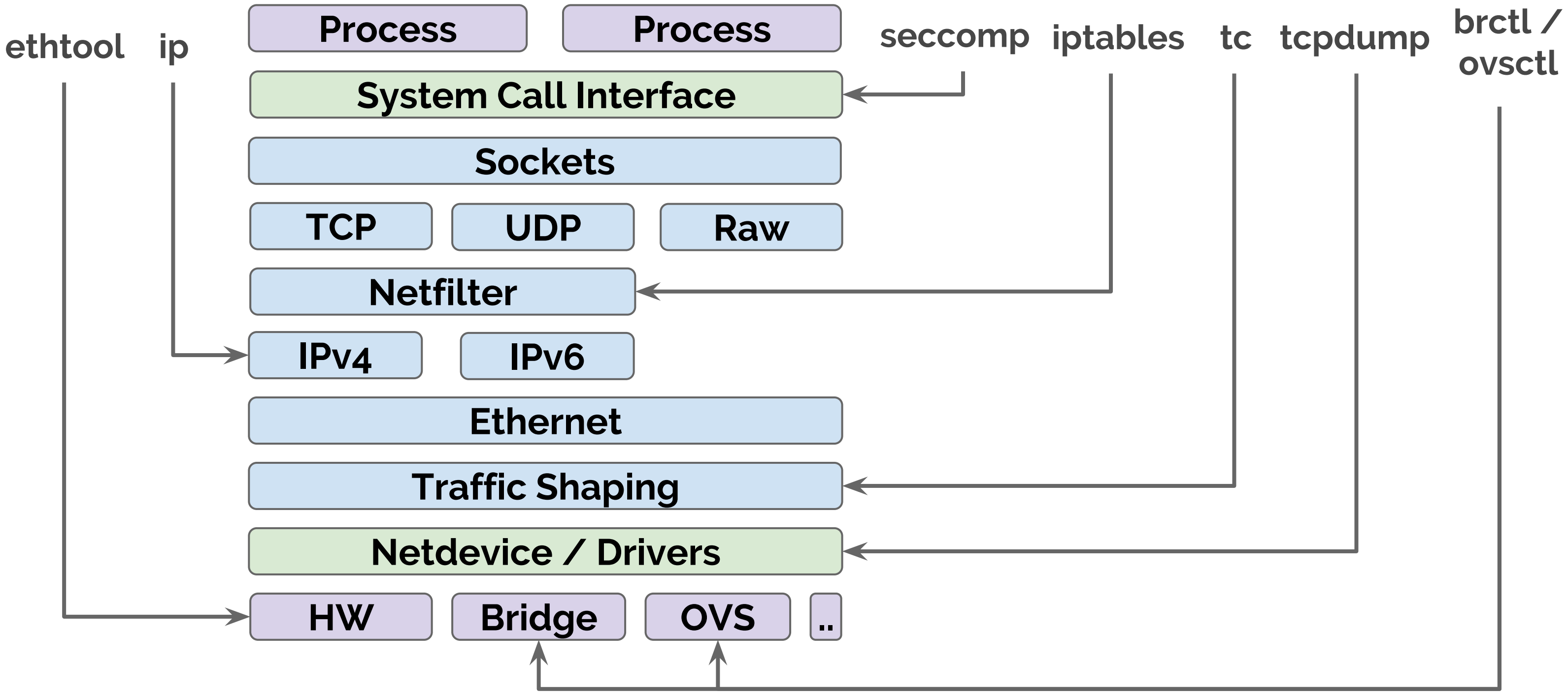
Pros:

- Strong userspace API compatibility guarantee. A 20 years old binary still works.
- Majority of Linux source code is not hardware specific.

Cons:

- Every layer pays the cost of the layers above and below.
- Very hard to bypass layers.

Problem #2: Per subsystem APIs



Problem #3: Development Process

The Good:

- Open and transparent process
- Excellent code quality
- Stability
- Available everywhere
- Almost entirely vendor neutral

The Bad:

- Hard to change
- Shouting is involved (getting better)
- Large and complicated codebase
- Upstreaming code is hard, consensus has to be found.
- Upstreaming is time consuming
- Depending on the Linux distribution, merged code can take years to become generally available
- Everybody maintains forks with 100-1000s backports

Problem #4: What is a container?

What the kernel knows about:

- **Processes & thread groups**
- **Cgroups**
  - Limits and accounting of CPU, memory, network, … Configured by container runtime.
- **Namespaces**
  - Isolation of process, CPU, mount, user, network, IPC, cgroup, UTS (hostname). Configured by container
  - runtime
- **IP addresses & port numbers**
  - Configured by container networking
- **System calls made & SELinux context**
  - Optionally configured by container runtime

What the kernel does not know:

- **Containers or Kubernetes pods**
  - There is no container ID in the kernel
- **Exposure requirements**
  - The kernel no longer knows whether an application should be exposed outside of the host or not.
- **API calls made between containers/pods**
  - Awareness stops at layer 4 (ports). While SELinux can control IPC, it can't control service to service API calls.
- **Servicemesh, huh?**

What now? Alternatives?

Give user space access to hardware

Expose the hardware directly to user space. It will be fine.

Examples: DPDK, UDMA, ..

Unikernel

Linus was wrong. The app should provide its own OS.

Examples: ClickOS, MirageOS, Rumprun, ...

Move OS to Userspace

We don't need kernel mode for most of the logic. Build on top of a minimal Linux.

Examples: User mode Linux, gVisor, ...

Rewrite Everything?

Total Estimated Cost to Develop Linux (average salary = **$75,662.08**/year, overhead = 2.40). **$1,372,340,206**

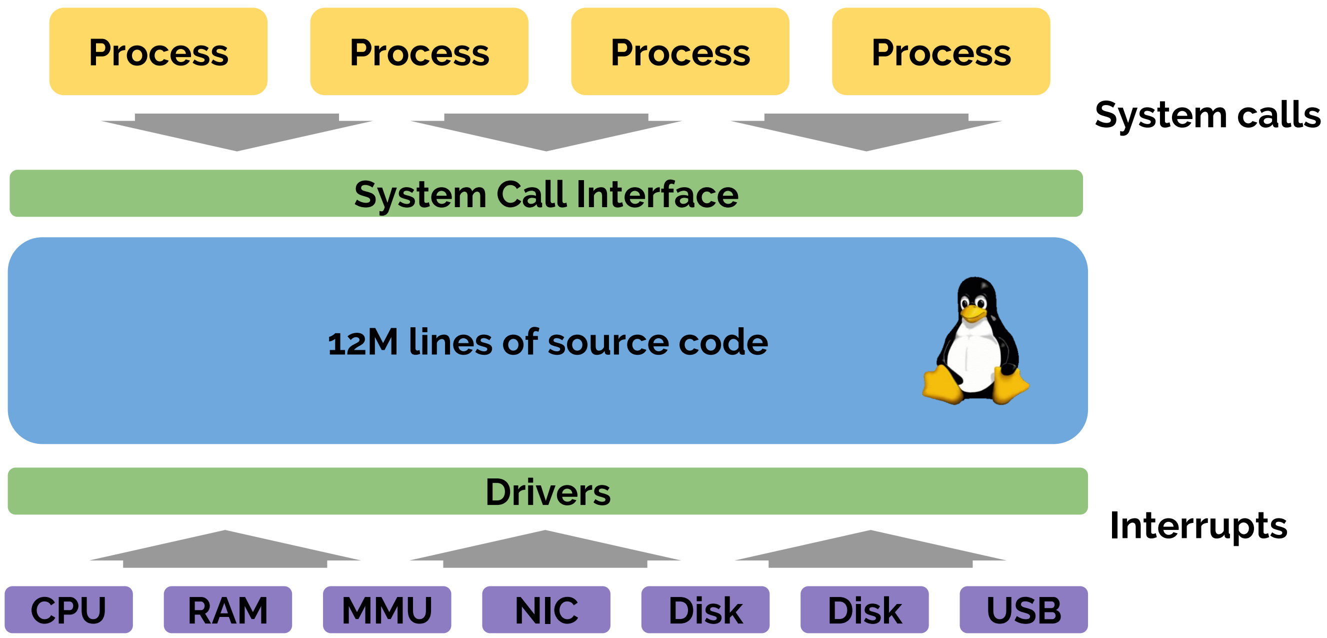
What is BPF?

Highly efficient sandboxed virtual machine in the Linux kernel making the Linux kernel programmable at native execution speed.

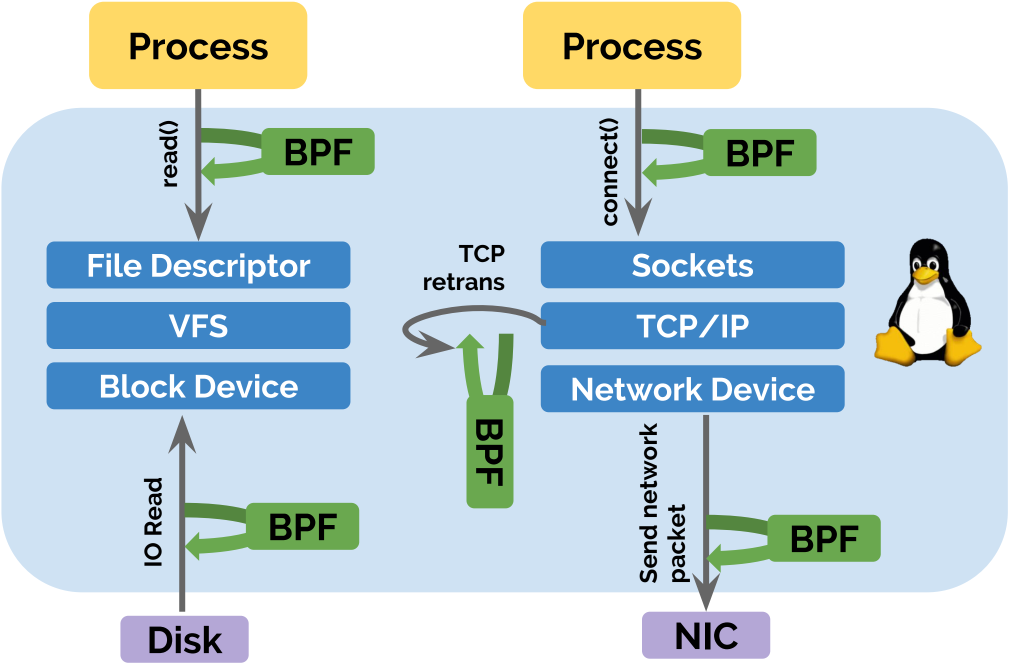
Jointly maintained by Cilium and Facebook with collaborations from Google, Red Hat, Netflix, Netronome, and many others.

```
$ clang -target bpf -emit-llvm -S \
   32-bit-example.c
$ llc -march=bpf 32-bit-example.ll
$ cat 32-bit-example.s
    cal:
      r1 = *(u32 *)(r1 + 0)
      r2 = *(u32 *)(r2 + 0)
      r2 += r1
      *(u32 *)(r3 + 0) = r2
      exit
```

The Linux kernel is event driven



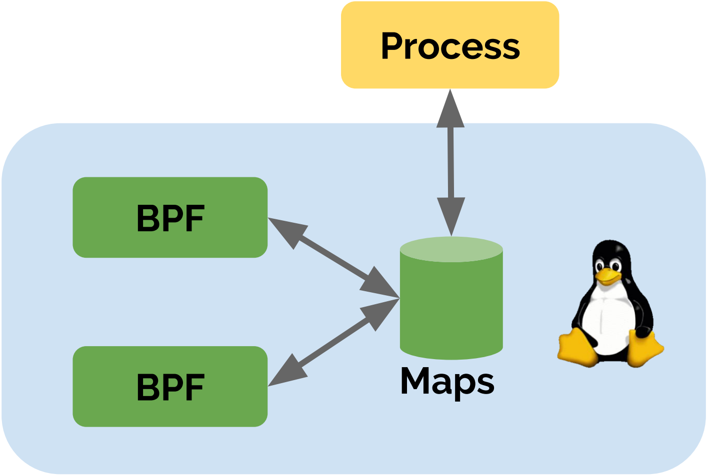
Run BPF program on event



Attachment points

- Kernel functions (kprobes)
- Userspace functions (uprobe)
- System calls
- Tracepoints
- Network devices (packet level)
- Sockets (data level)
- Network device (DMA level) [XDP]
- ...

BPF Maps



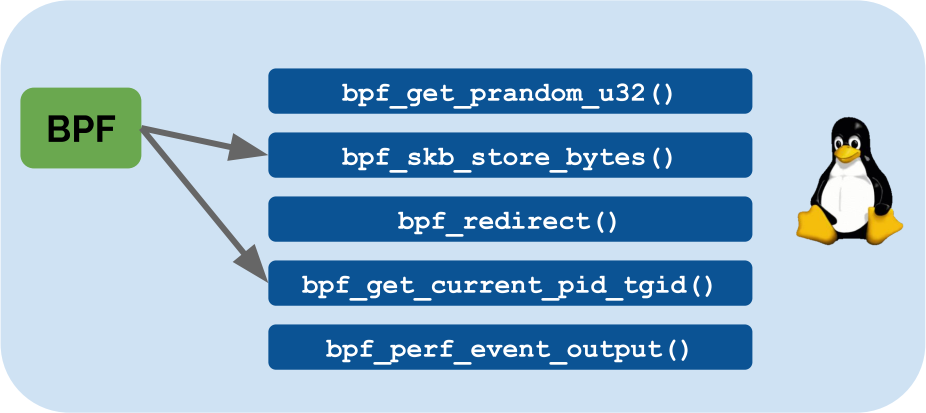
BPF map use cases:

- Hold program state
- Share state between programs
- Share state with user space
- Export metrics & statistics
- Configure programs

Map types:

- Hash tables
- Arrays
- LRU (Least recently used)
- Ring buffer
- Stack trace
- LPM (Longest prefix match)

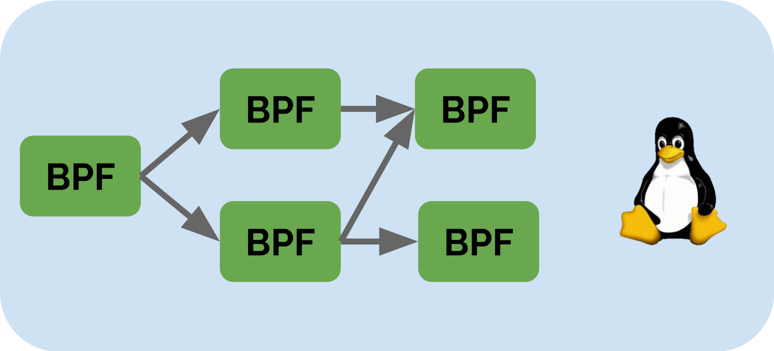
BPF Helpers



BPF helpers:

- Stable kernel API exposed to BPF programs to interact with the kernel
- Includes ability to:
  - Get process/cgroup context
  - Manipulate network packets and forwarding
  - Access BPF maps
  - Access socket data
  - Send metrics to user space
  - ...

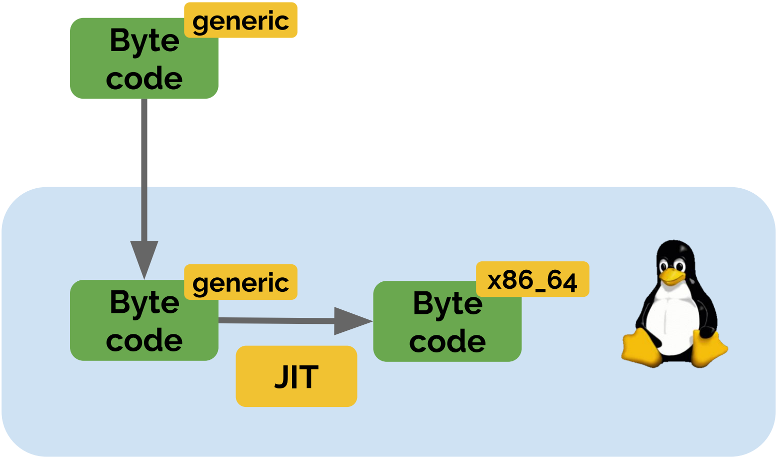
BPF Tail Calls



BPF tail calls:

- Chain logical programs together
- Implement function calls
- Must be within same program type

BPF JIT Compiler



JIT Compiler

- Ensures native execution performance without requiring to understand CPU
- Compiles BPF bytecode to CPU architecture specific instruction set

Supported architectures:

- X86_64, arm64, ppc64, s390x, mips64, sparc64, arm

BPF Contributors

380 Daniel Borkmann (Cilium, Maintainer)
161 Alexei Starovoitov (Facebook, Maintainer)
160 Jakub Kicinski Netronome
110 John Fastabend (Cilium)
96 Yonghong Song (Facebook)
95 Martin KaFai Lau (Facebook)
94 Jesper Dangaard Brouer (Red Hat)
74 Quentin Monnet (Netronome)
45 Roman Gushchin (Facebook)
45 Andrey Ignatov (Facebook)

Top contributors of the total 186 contributors to BPF from January 2016 to November 2018.

BPF Use Cases

- L3-L4 Load balancing
- Network security
- Traffic optimization
- Profiling

https://code.fb.com/open-source/linux/

- Replacing iptables with BPF (bpfilter)
- NFV & Load balancing (XDP)
- Profiling & Tracing

- QoS & Traffic optimization
- Network Security
- Profiling

- Performance Troubleshooting
- Tracing & Systems Monitoring
- Networking

Simple Kprobe Example

Example: BPF program using gobpf/bcc:

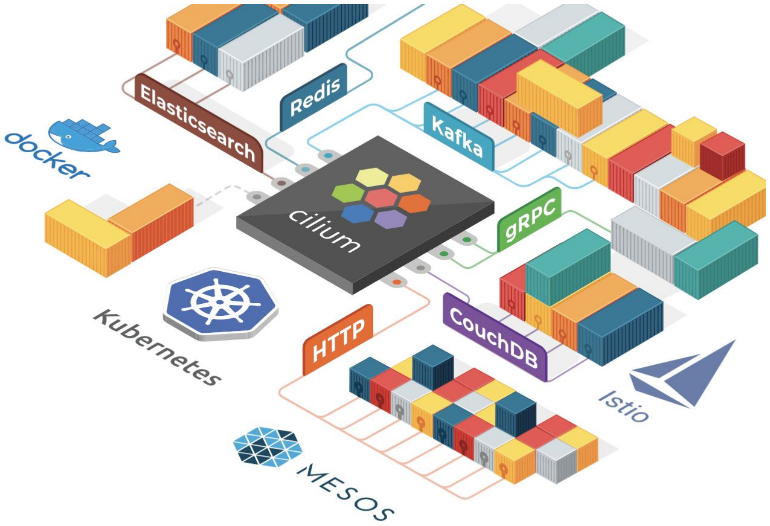
```
int syscall__ret_execve(struct pt_regs *ctx)
{
        struct comm_event event = {
                .pid = bpf_get_current_pid_tgid() >> 32,
                .type = TYPE_RETURN,
        };

        bpf_get_current_comm(&event.comm, sizeof(event.comm));
        comm_events.perf_submit(ctx, &event, sizeof(event));

        return 0;
}
```

What is Cilium?

Cilium is open source software for transparently providing and securing the network and API connectivity between application services deployed using Linux container management platforms like Kubernetes, Docker, and Mesos.

At the foundation of Cilium is the new Linux kernel technology BPF, which enables the dynamic insertion of powerful security, visibility, and networking control logic within Linux itself. Besides providing traditional network level security, the flexibility of BPF enables security on API and process level to secure communication within a container or pod.

[Read More]



Project Goals

Approachable BPF

- Make the efficiency and flexibility of BPF available in an approachable way
- Automate program creation and management
- Provide an extendable platform

Microservices-aware Linux

- Use the flexibility of BPF to make the Linux kernel aware of cloud native concepts such as containers and APIs.

Security

- Use the additional visibility of BPF to provide security for microservices including:
  - API awareness
  - Identity based enforcement
  - Process level context enforcement

Performance

- Leverage the execution performance and JIT compiler to provide a highly efficient implementation.

Cilium Use Cases

Container Networking

- Highly efficient and flexible networking
- CNI and CMM plugins
- IPv4, IPv6, NAT46, direct routing, encapsulation
- Multi cluster routing

Microservices Security

- Identity-based L3-L4 network security
- Accelerated API-aware security via Envoy (HTTP, gRPC, Kafka, Cassandra, memcached, ..)
- DNS aware policies
- SSL data visibility via kTLS

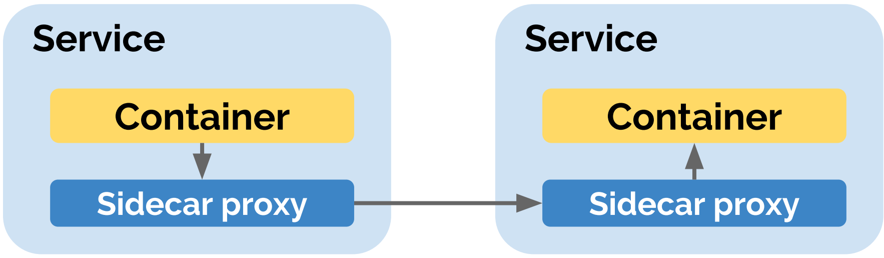
Service Load balancing:

- Highly scalable L3-L4 load balancing implementation
- Kubernetes service implementation or API driven.

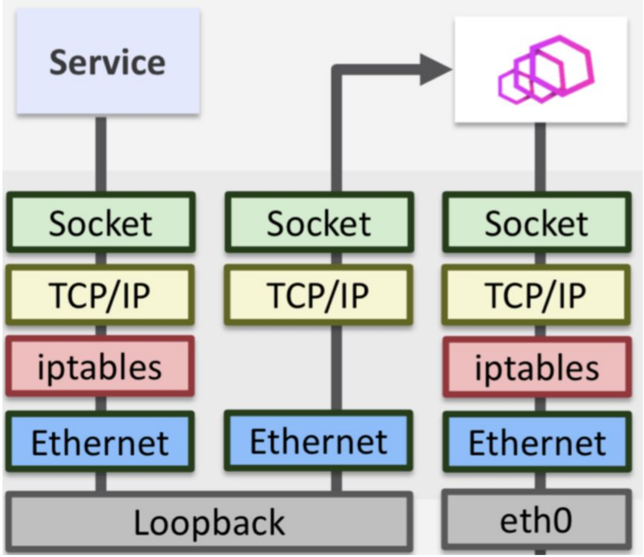
Servicemesh acceleration:

- Minimize overhead when injecting servicemesh sidecar proxies

BPF-based servicemesh
Acceleration

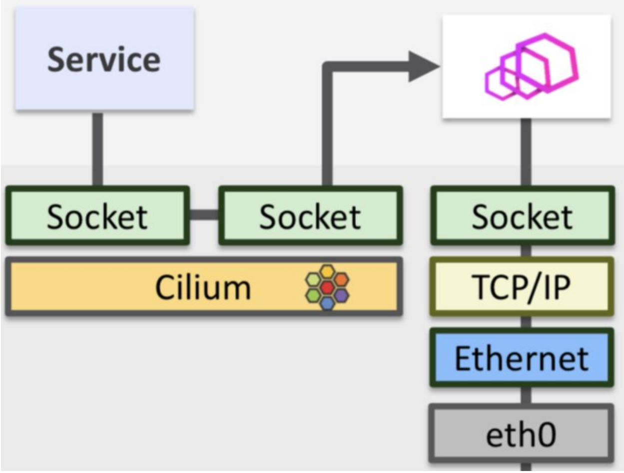


How it really looks:

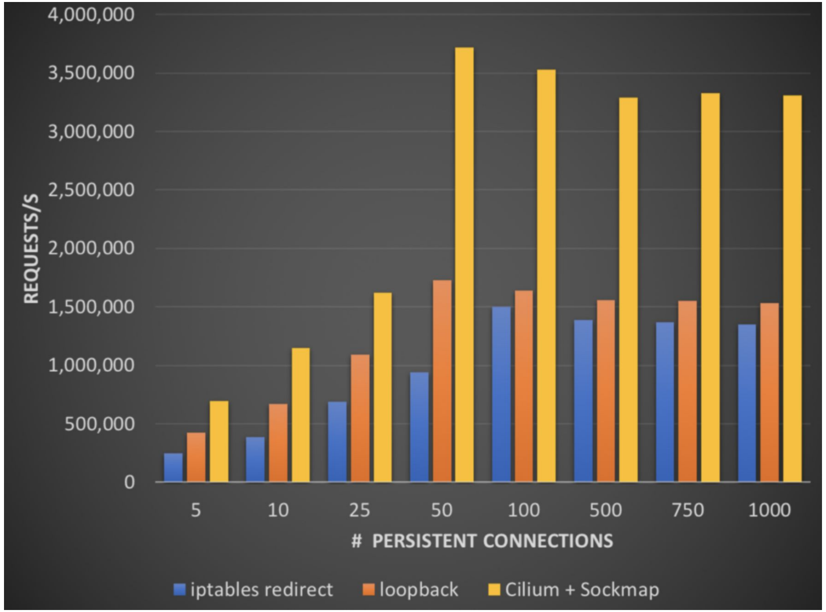


BPF-based servicemesh
Acceleration

Accelerate the service to sidecar communication



~3.5x performance improvement



Other BPF projects

cilium

Tracing / Profiling:

- BPFTrace - DTrace for Linux (Brendan Gregg, et al.)
- bpfd - Load BPF programs into entire clusters (Joel Fernandes, Google)

Frameworks:

- gobpf - Go based framework to write BPF programs
- BCC - Python framework to write BPF programs

Load balancing:

- Katran - Source code of Facebook's primary L3-L4 LB (Facebook team)

Security:

- Seccomp - Advanced BPF version of Seccomp (Kernel team)

DDoS mitigation:

- bpftools - DDOS mitigation tool with iptables like syntax (Cloudflare)

... and many more

Thank you!

Source Code:
https://github.com/cilium/cilium
BPF reference guide:
http://docs.cilium.io/en/stable/bpf/

Twitter:
@[ciliumproject](https://twitter.com/ciliumproject)
Website:
https://cilium.io/

cilium