



Understanding (and Troubleshooting) the eBPF Datapath in Cilium

Nate Sweet - Senior Software Engineer - DigitalOcean

Full Slides @ <https://www.nathansweet.me/assets/kubecon-na-19-slides>

eBPF Libs:

- github.com/newtools/ebpf
- github.com/cilium/ebpf (WIP)



Why should we care?

- The network is the bottleneck
- eBPF is Spreading
- [Metrics are Money](#)



What is eBPF?

```
#define KBUILD_MODNAME "xdp_dummy"
#include <uapi/linux/bpf.h>
#include <linux/if_ether.h>
#include "bpf_helpers.h"
```

```
struct bpf_elf_map SEC("maps") blacklist = {
    .type      = BPF_MAP_TYPE_HASH,
    .size_key  = sizeof(u32),
    .size_value = sizeof(u8),
    .max_elem  = 100000,
};
```

```
struct arp_t {
    unsigned short    htype;
    unsigned short    ptype;
    unsigned char     hlen;
    unsigned char     plen;
    unsigned short    oper;
    unsigned long long sha:48;
    unsigned long long spa:32;
    unsigned long long tha:48;
    unsigned int      tpa;
} __attribute__((packed));
```

```
SEC("drop_bl_arp")
int drop_bl_arp(struct xdp_md *ctx) {
    void *data_end = (void *) (long) ctx->data_end;
    void *data = (void *) (long) ctx->data;
    u32 ip_src;
    u64 *value;
    struct ethhdr *eth = data;

    if (eth->h_proto != htons(0x0806)) {
        return XDP_PASS;
    }

    struct arp_t *arp = data + sizeof(*eth);

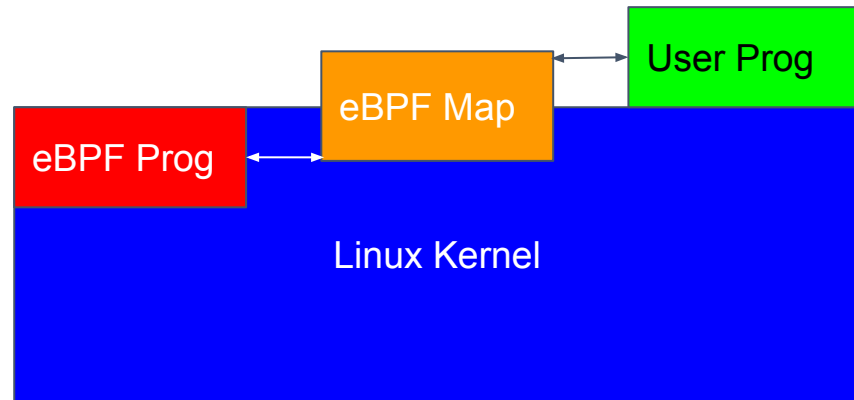
    ip_src = arp->tpa;
    value = bpf_map_lookup_elem(&blacklist,
    &ip_src);
    if (value) {
        return XDP_DROP;
    }

    return XDP_PASS;
}
```



Why eBPF?

- eBPF is fast
- eBPF is flexible
- eBPF separates data from functionality





A Brief History

- Steven McCanne, et al, in 1993 - *The BSD Packet Filter*
- Jeffrey C. Mogul, et al, in 1987 - first open source implementation of a packet filter.

2.1. Historical background

As far as we are aware, the idea (and name) of the packet filter was first proposed by David Boggs and Edward Taft at Xerox Alto [3]. Because the Alto operating system supported multiple processes, and because security was not important, the filter was implemented in user level programs; these procedures were called by the packet filter. The Unix implementation of the packet filter was done in 1980.

- [3] David Boggs and Edward Taft.
Private communication.
1987.



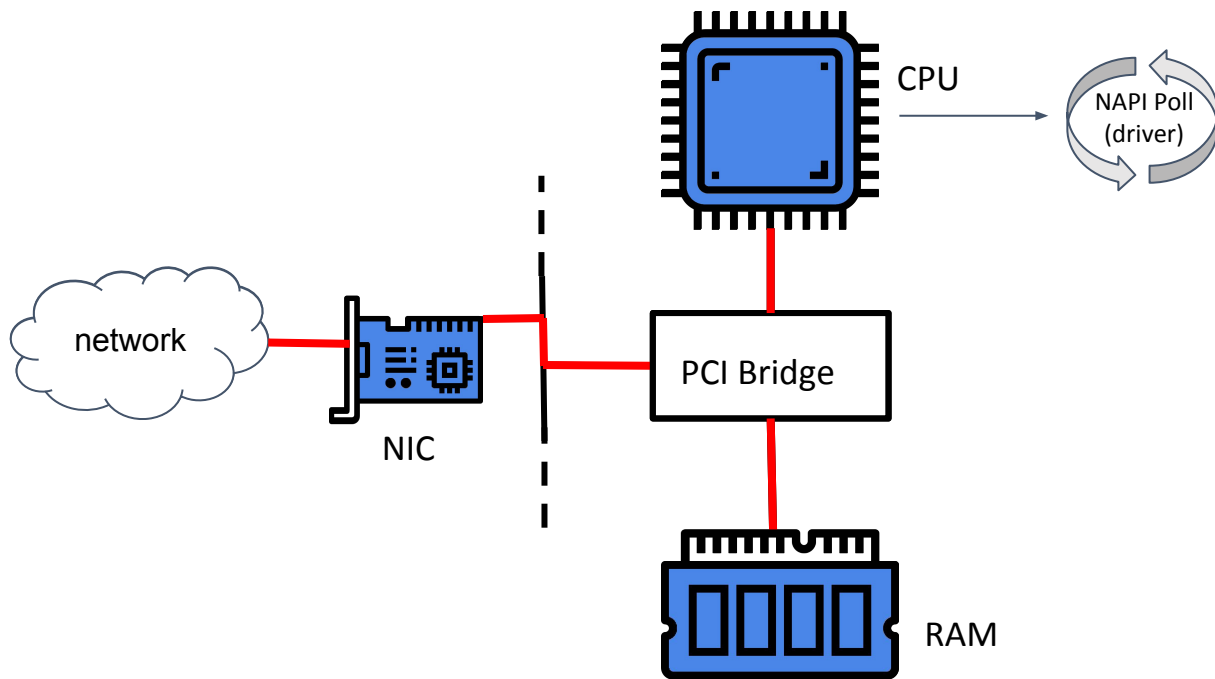


What is Cilium, and why should we care?

- eBPF is hard to write
- Cilium is...ugh...easy(er)?
- Cilium is a CNI and replacement for kube-proxy

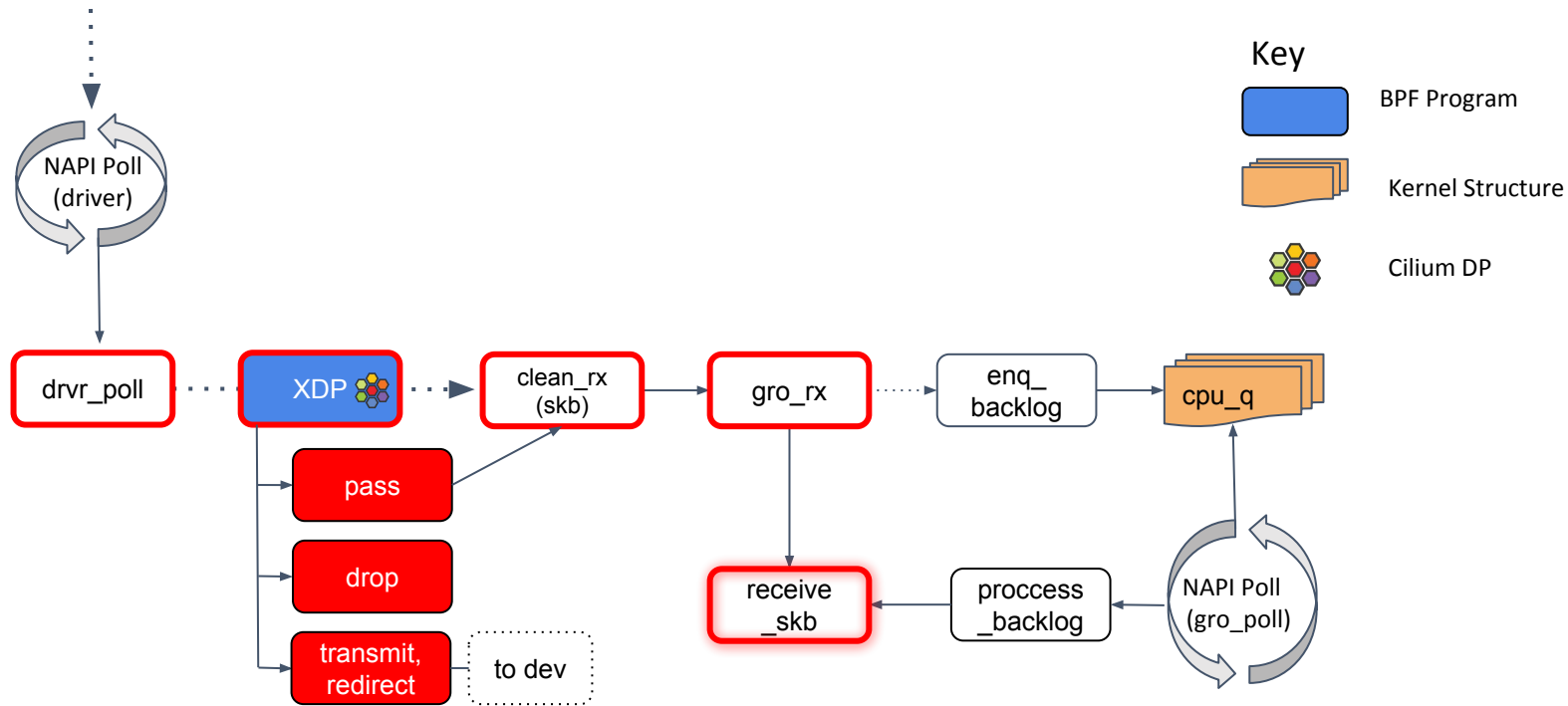


The Default Network Datapath, Layer 1 - 2

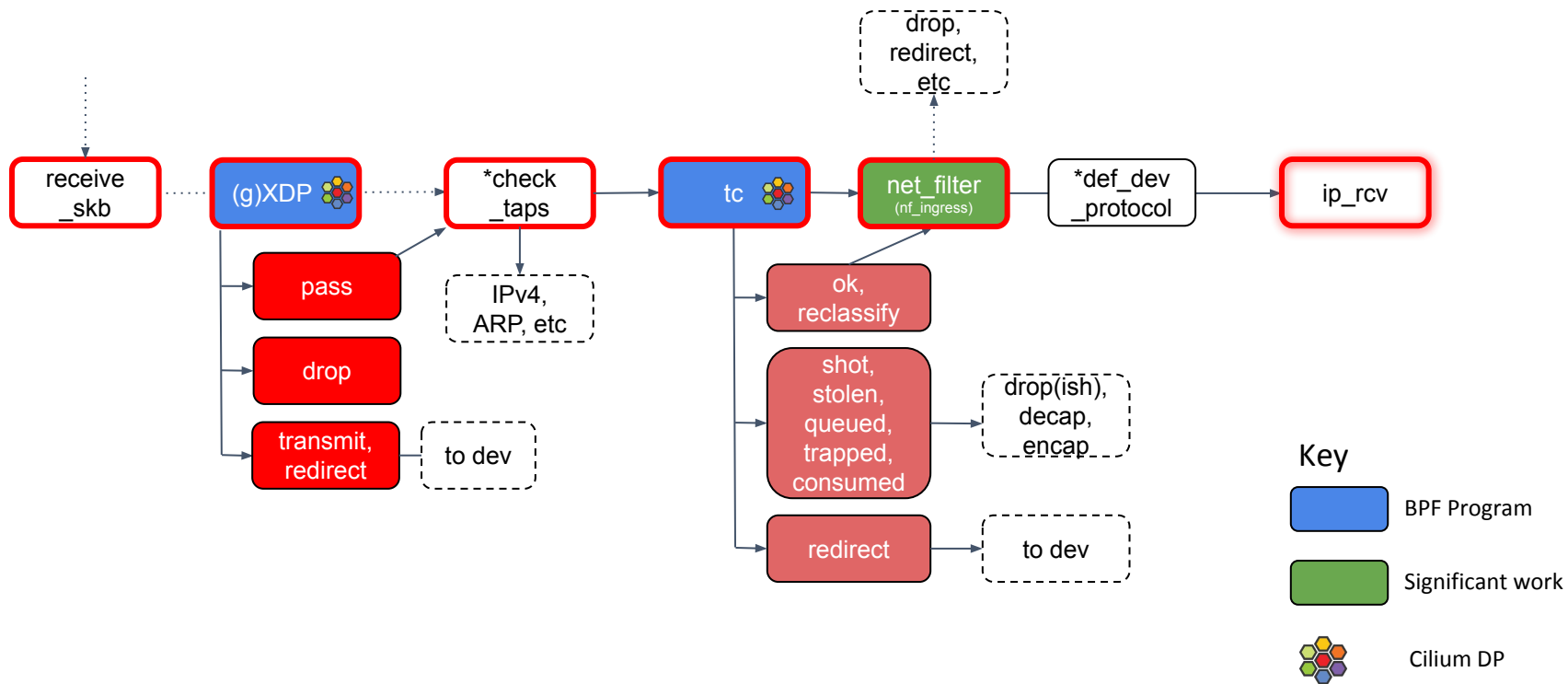




The Default Network Datapath, Layer 2 continued

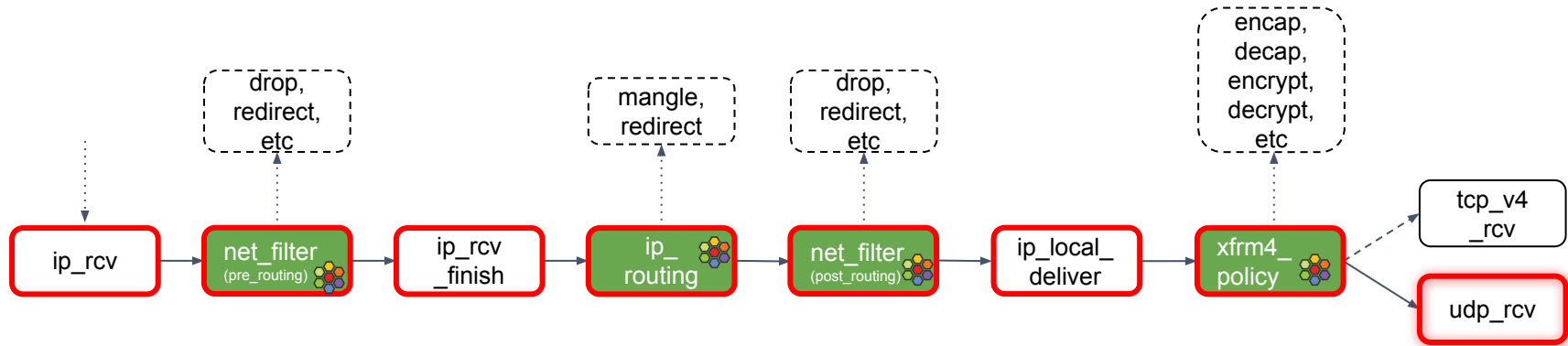


The Default Network Datapath, Layer 2 - 3

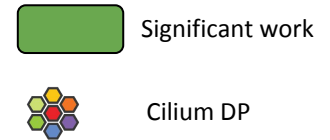


*No such function

The Default Network Datapath, Layer 3 - 4

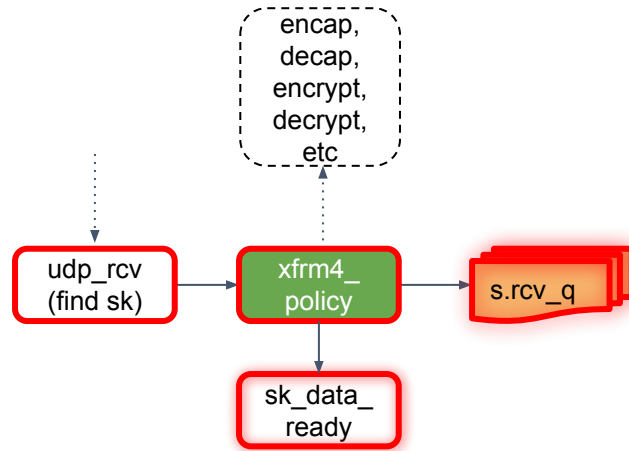


Key





The Default Network Datapath, Layer 4



Key



Significant Work



Kernel Structure



The diagram illustrates the Cilium DP flow for handling network packets. It starts with a **rcv_q** (kernel structure) which feeds into **udp_recv_msg** (BPF program). This BPF program then interacts with the **cgroup** (BPF program), which in turn interacts with **sock_ops** (BPF program). The **sock_ops** BPF program then interacts with the **user space** (cloud). The **sock_ops** BPF program also branches into four actions: **drop**, **forward**, **redirect**, and **do_nothing** (all BPF programs). The **do_nothing** action triggers a **NAPI Poll (epoll)** loop, which then feeds back into the **do_epoll_wait** BPF program, which then interacts with the **user space** cloud.

Key:

- BPF Program** (Blue rectangle)
- Kernel Structure** (Orange rectangle)
- Cilium DP** (Hexagon icon)



Kubernetes - Cilium - Kernel

Kubernetes	Cilium	Kernel
Endpoint (includes Pods)	Endpoint	tc, cgroup socket BPF, sock_ops BPF, XDP
Network Policy	Cilium Network Policy	XDP, tc, sock-ops
Service (node ports, cluster ips, etc)	Service	XDP, tc
Node	Node	ip-xfrm (for encryption), ip tables for initial decapsulation routing (if vxlan), veth-pair



Demo



Sources and Thanks

History

- Steven McCanne and Van Jacobson, *The BSD Packet Filter: A New Architecture for User-level Packet Capture*, 1992, <https://www.tcpdump.org/papers/bpf-usenix93.pdf>
- Jeffrey C Mogul, *The Packet Filter: An Efficient Mechanism for User-level Network Code*, 1987, <https://www.hpl.hp.com/techreports/Compaq-DEC/WRL-87-2.pdf>

Linux Datapath

- Linux Source Code
- Packagecloud, *Monitoring and Tuning the Linux Networking Stack: Receiving Data*, 2016, <https://blog.packagecloud.io/enq/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#napi-poll-function-and-weight> - this is a great resource, as well as its counterpart: *Monitoring and Tuning the Linux Networking Stack: Sending Data*, <https://blog.packagecloud.io/enq/2017/02/06/monitoring-tuning-linux-networking-stack-sending-data/>

BPF and Userspace Programs

- Linux Manual Pages for “tc-bpf”: <http://man7.org/linux/man-pages/man8/tc-bpf.8.html>
- Linux Manual Pages for “packet” <http://man7.org/linux/man-pages/man7/packet.7.html>
- Linux Manual Pages for “ip-xfrm” <http://man7.org/linux/man-pages/man8/ip-xfrm.8.html>
- Linux Manual Pages for “socket” <http://man7.org/linux/man-pages/man2/socket.2.html>
- Linux Manual Pages for “epoll” <http://man7.org/linux/man-pages/man7/epoll.7.html>

Cilium and Kubernetes

- Cilium Source Code
- Cilium Docs and Architecture Diagrams, <https://cilium.readthedocs.io/en/stable/architecture/>
- Kubernetes Documentation on Networking, <https://kubernetes.io/docs/concepts/>

Thanks

To my team at DigitalOcean, and my friends at 100 State, both of whom listened to this talk.