BPF Programmable Netdevice

Daniel Borkmann (Isovalent)

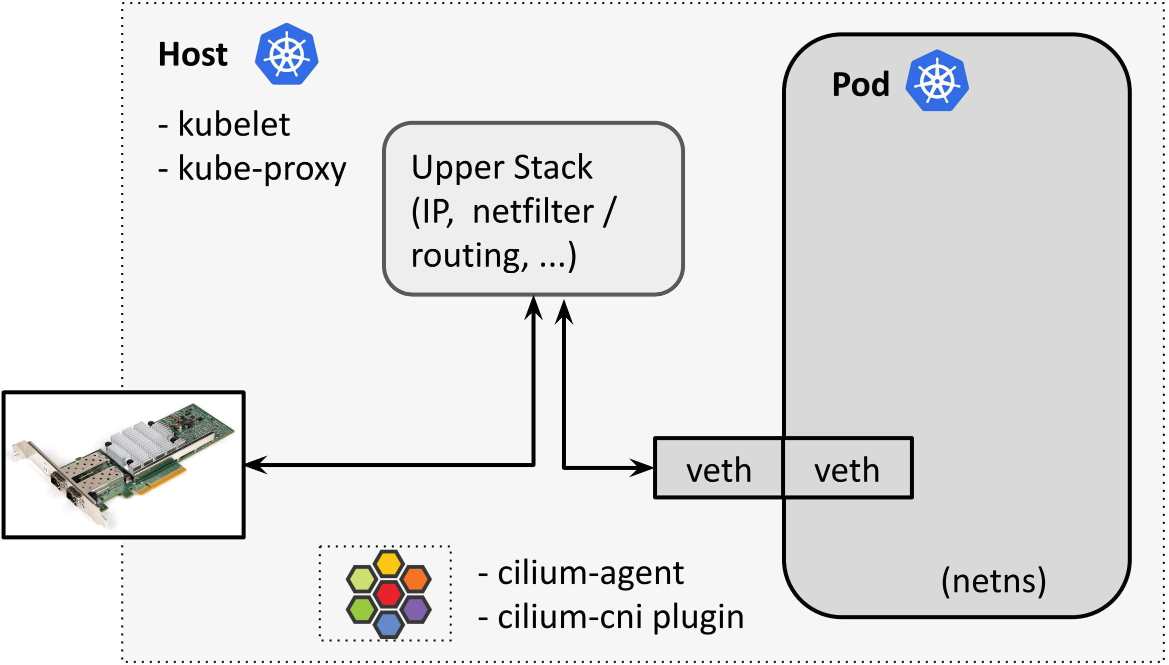
LPC Networking & BPF Track 2023

Goal for this talk: Can we achieve lowest possible networking overhead for K8s Pods?

Standard K8s Datapath Architecture:

Problems:

- kube-proxy scalability
- Routing via upper stack
- Potential reasons:
  - Cannot replace kube-proxy
  - Custom netfilter rules
  - Just “went with defaults”



Host

- kubelet
- kube-proxy

Upper Stack (IP, netfilter / routing, ...)

Pod

veth

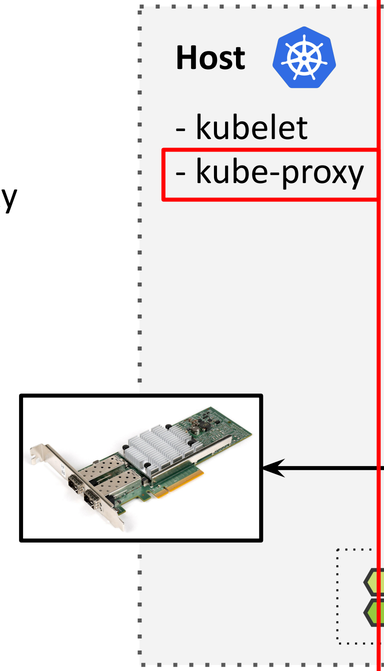
veth

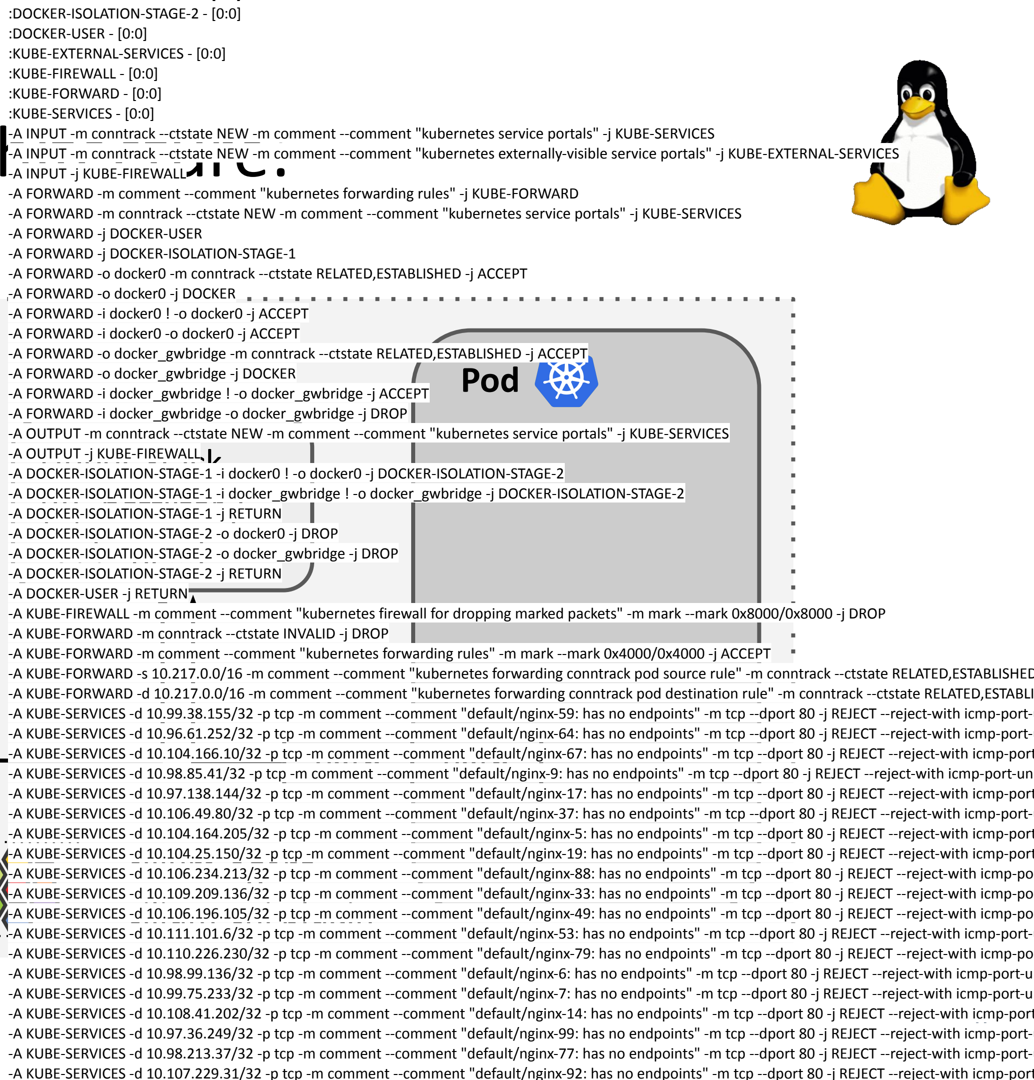
(netns)

- cilium-agent
- cilium-cni plugin

Standard K8s Datapath Arc

Problems:
- kube-proxy scalability
- Routing via upper stack
- Potential reasons:
  - Cannot replace kube-proxy
  - Custom netfilter rules
  - Just “went with defaults”



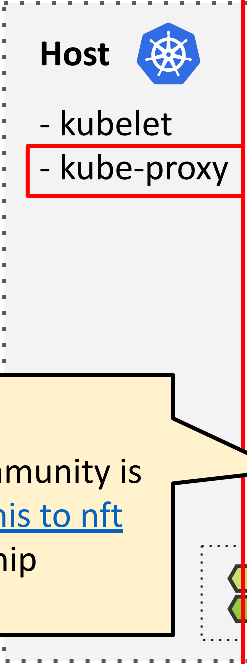


```
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
:KUBE-EXTERNAL-SERVICES - [0:0]
:KUBE-FIREWALL - [0:0]
:KUBE-FORWARD - [0:0]
:KUBE-SERVICES - [0:0]
-A INPUT -m conntrack --ctstate NEW -m comment --comment "kubernetes service portals" -j KUBE-SERVICES
-A INPUT -m conntrack --ctstate NEW -m comment --comment "kubernetes externally-visible service portals" -j KUBE-EXTERNAL-SERVICES
-A INPUT -j KUBE-FIREWALL
-A FORWARD -m comment --comment "kubernetes forwarding rules" -j KUBE-FORWARD
-A FORWARD -m conntrack --ctstate NEW -m comment --comment "kubernetes service portals" -j KUBE-SERVICES
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A FORWARD -o docker_gwbridge -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker_gwbridge -j DOCKER
-A FORWARD -i docker_gwbridge ! -o docker_gwbridge -j ACCEPT
-A FORWARD -i docker_gwbridge -o docker_gwbridge -j DROP
-A OUTPUT -m conntrack --ctstate NEW -m comment --comment "kubernetes service portals" -j KUBE-SERVICES
-A OUTPUT -j KUBE-FIREWALL
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -i docker_gwbridge ! -o docker_gwbridge -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -o docker_gwbridge -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
-A KUBE-FIREWALL -m comment --comment "kubernetes firewall for dropping marked packets" -m mark --mark 0x8000/0x8000 -j DROP
-A KUBE-FORWARD -m conntrack --ctstate INVALID -j DROP
-A KUBE-FORWARD -m comment --comment "kubernetes forwarding rules" -m mark --mark 0x4000/0x4000 -j ACCEPT
-A KUBE-FORWARD -s 10.217.0.0/16 -m comment --comment "kubernetes forwarding conntrack pod source rule" -m conntrack --ctstate RELATED,ESTABLISHED
-A KUBE-FORWARD -d 10.217.0.0/16 -m comment --comment "kubernetes forwarding conntrack pod destination rule" -m conntrack --ctstate RELATED,ESTABLI
-A KUBE-SERVICES -d 10.99.38.155/32 -p tcp -m comment --comment "default/nginx-59: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.96.61.252/32 -p tcp -m comment --comment "default/nginx-64: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.104.166.10/32 -p tcp -m comment --comment "default/nginx-67: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.98.85.41/32 -p tcp -m comment --comment "default/nginx-9: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-un
-A KUBE-SERVICES -d 10.97.138.144/32 -p tcp -m comment --comment "default/nginx-17: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.106.49.80/32 -p tcp -m comment --comment "default/nginx-37: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.104.164.205/32 -p tcp -m comment --comment "default/nginx-5: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.104.25.150/32 -p tcp -m comment --comment "default/nginx-19: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.106.234.213/32 -p tcp -m comment --comment "default/nginx-88: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-po
-A KUBE-SERVICES -d 10.109.209.136/32 -p tcp -m comment --comment "default/nginx-33: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-po
-A KUBE-SERVICES -d 10.106.196.105/32 -p tcp -m comment --comment "default/nginx-49: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-po
-A KUBE-SERVICES -d 10.111.101.6/32 -p tcp -m comment --comment "default/nginx-53: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.110.226.230/32 -p tcp -m comment --comment "default/nginx-79: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-po
-A KUBE-SERVICES -d 10.98.99.136/32 -p tcp -m comment --comment "default/nginx-6: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-u
-A KUBE-SERVICES -d 10.99.75.233/32 -p tcp -m comment --comment "default/nginx-7: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-u
-A KUBE-SERVICES -d 10.108.41.202/32 -p tcp -m comment --comment "default/nginx-14: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.97.36.249/32 -p tcp -m comment --comment "default/nginx-99: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.98.213.37/32 -p tcp -m comment --comment "default/nginx-77: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.107.229.31/32 -p tcp -m comment --comment "default/nginx-92: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
```

Standard K8s Datapath Architecture

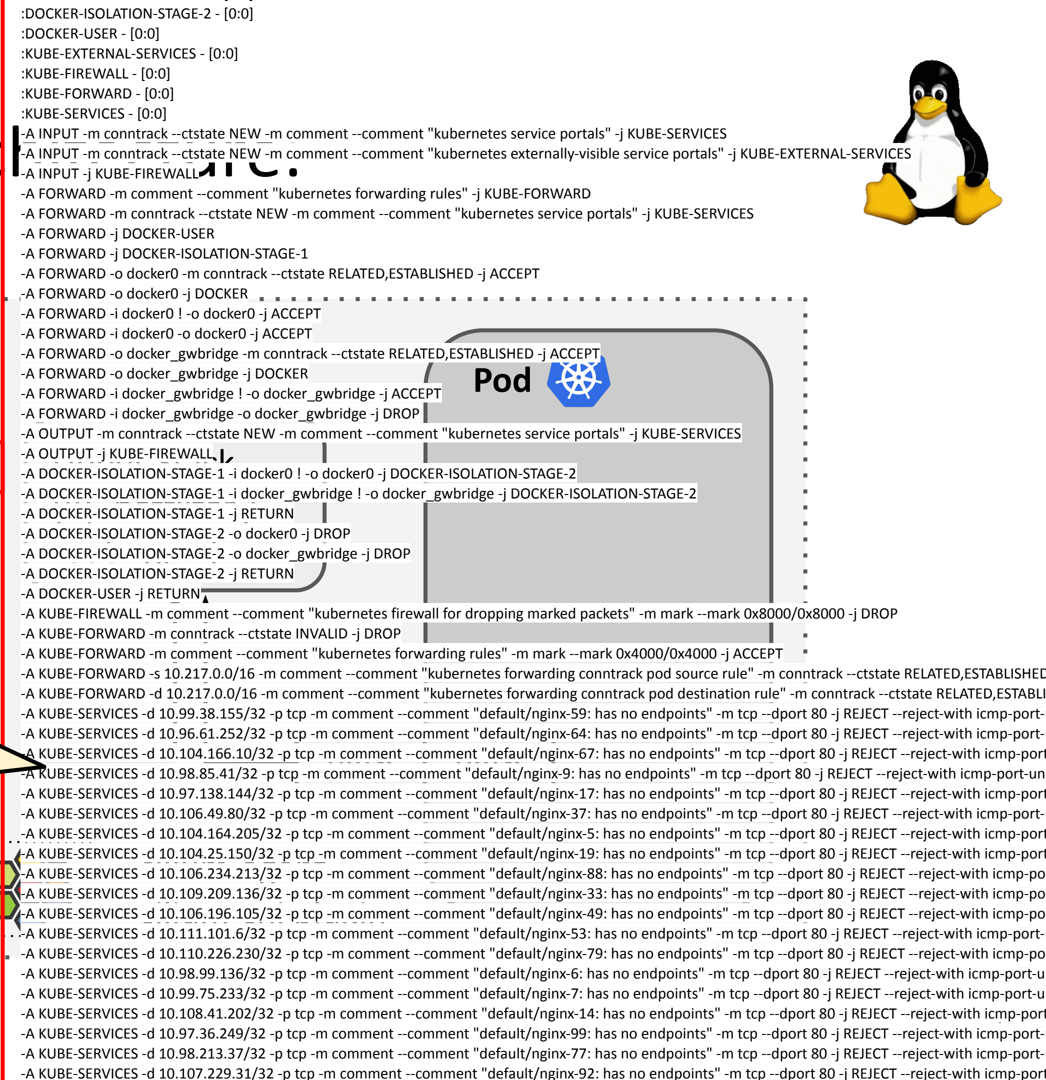
Problems:

- kube-proxy scalability
- Routing via upper stack
- Potential reasons:
  - Cannot replace kube-proxy
  - Custom netfilter rules
  - Just “went with defaults”



KubeCon ‘23:

K8s sig-networking community is now starting to [move this to nft](#) since RHEL10 will not ship iptables anymore.

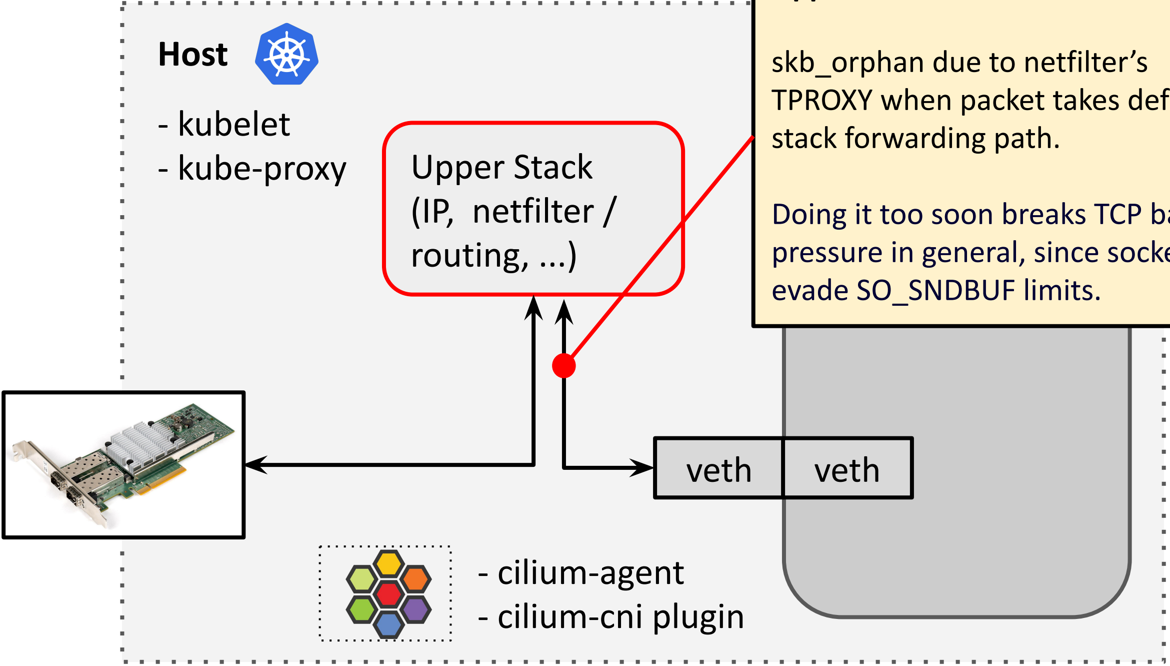


```
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
:KUBE-EXTERNAL-SERVICES - [0:0]
:KUBE-FIREWALL - [0:0]
:KUBE-FORWARD - [0:0]
:KUBE-SERVICES - [0:0]
-A INPUT -m conntrack --ctstate NEW -m comment --comment "kubernetes service portals" -j KUBE-SERVICES
-A INPUT -m conntrack --ctstate NEW -m comment --comment "kubernetes externally-visible service portals" -j KUBE-EXTERNAL-SERVICES
-A INPUT -j KUBE-FIREWALL
-A FORWARD -m comment --comment "kubernetes forwarding rules" -j KUBE-FORWARD
-A FORWARD -m conntrack --ctstate NEW -m comment --comment "kubernetes service portals" -j KUBE-SERVICES
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A FORWARD -o docker_gwbridge -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker_gwbridge -j DOCKER
-A FORWARD -i docker_gwbridge ! -o docker_gwbridge -j ACCEPT
-A FORWARD -i docker_gwbridge -o docker_gwbridge -j DROP
-A OUTPUT -m conntrack --ctstate NEW -m comment --comment "kubernetes service portals" -j KUBE-SERVICES
-A OUTPUT -j KUBE-FIREWALL
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -i docker_gwbridge ! -o docker_gwbridge -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -o docker_gwbridge -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
-A KUBE-FIREWALL -m comment --comment "kubernetes firewall for dropping marked packets" -m mark --mark 0x8000/0x8000 -j DROP
-A KUBE-FORWARD -m conntrack --ctstate INVALID -j DROP
-A KUBE-FORWARD -m comment --comment "kubernetes forwarding rules" -m mark --mark 0x4000/0x4000 -j ACCEPT
-A KUBE-FORWARD -s 10.217.0.0/16 -m comment --comment "kubernetes forwarding conntrack pod source rule" -m conntrack --ctstate RELATED,ESTABLISHED
-A KUBE-FORWARD -d 10.217.0.0/16 -m comment --comment "kubernetes forwarding conntrack pod destination rule" -m conntrack --ctstate RELATED,ESTABLI
-A KUBE-SERVICES -d 10.99.38.155/32 -p tcp -m comment --comment "default/nginx-59: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.96.61.252/32 -p tcp -m comment --comment "default/nginx-64: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.104.166.10/32 -p tcp -m comment --comment "default/nginx-67: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.98.85.41/32 -p tcp -m comment --comment "default/nginx-9: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-un
-A KUBE-SERVICES -d 10.97.138.144/32 -p tcp -m comment --comment "default/nginx-17: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.106.49.80/32 -p tcp -m comment --comment "default/nginx-37: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.104.164.205/32 -p tcp -m comment --comment "default/nginx-5: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.104.25.150/32 -p tcp -m comment --comment "default/nginx-19: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.106.234.213/32 -p tcp -m comment --comment "default/nginx-88: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-po
-A KUBE-SERVICES -d 10.109.209.136/32 -p tcp -m comment --comment "default/nginx-33: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-po
-A KUBE-SERVICES -d 10.106.196.105/32 -p tcp -m comment --comment "default/nginx-49: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-po
-A KUBE-SERVICES -d 10.111.101.6/32 -p tcp -m comment --comment "default/nginx-53: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.110.226.230/32 -p tcp -m comment --comment "default/nginx-79: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-po
-A KUBE-SERVICES -d 10.98.99.136/32 -p tcp -m comment --comment "default/nginx-6: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-u
-A KUBE-SERVICES -d 10.99.75.233/32 -p tcp -m comment --comment "default/nginx-7: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-u
-A KUBE-SERVICES -d 10.108.41.202/32 -p tcp -m comment --comment "default/nginx-14: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
-A KUBE-SERVICES -d 10.97.36.249/32 -p tcp -m comment --comment "default/nginx-99: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.98.213.37/32 -p tcp -m comment --comment "default/nginx-77: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port-
-A KUBE-SERVICES -d 10.107.229.31/32 -p tcp -m comment --comment "default/nginx-92: has no endpoints" -m tcp --dport 80 -j REJECT --reject-with icmp-port
```

Standard K8s Datapath Architecture:

Problems:

- kube-proxy scalability
- Routing via upper stack
- Potential reasons:
  - Cannot replace kube-proxy
  - Custom netfilter rules
  - Just "went with defaults"



Upper Stack:

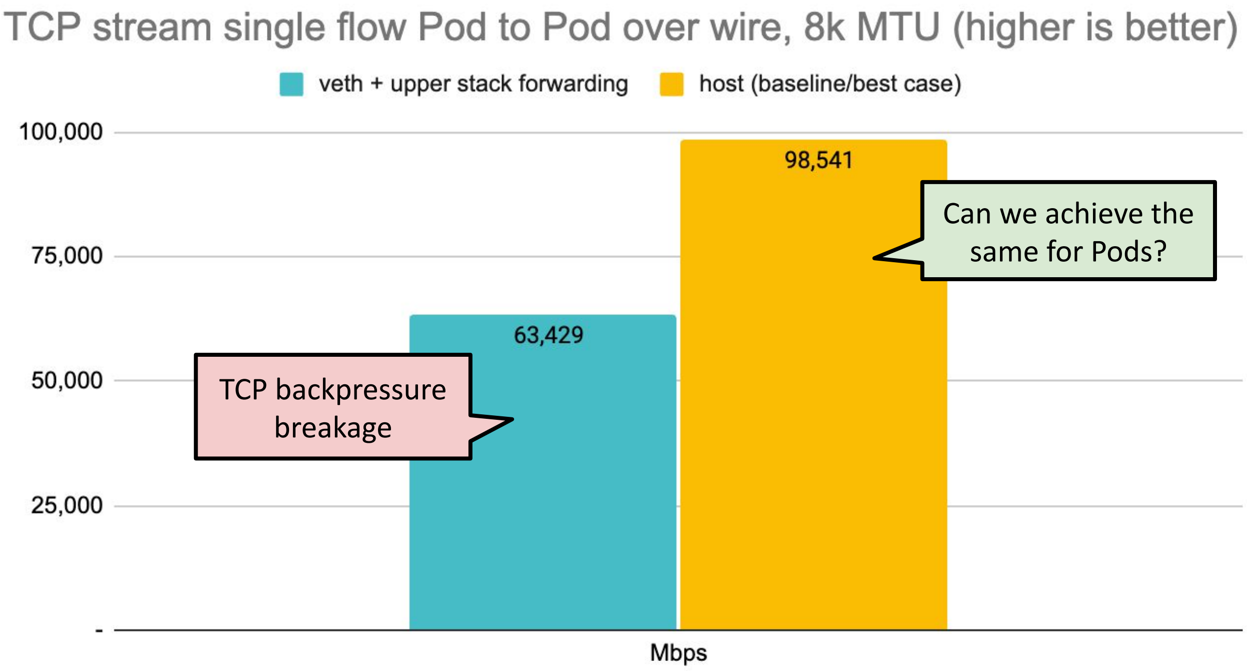
skb_orphan due to netfilter's TPROXY when packet takes default stack forwarding path.

Doing it too soon breaks TCP back pressure in general, since socket can evade SO_SNDBUF limits.

Standard K8s Datapath Architecture:

Problems:

- kube-proxy scalability
- Routing via upper stack
- Potential reasons:
  - Cannot replace kube-proxy
  - Custom netfilter rules
  - Just “went with defaults”



TCP stream single flow Pod to Pod over wire, 8k MTU (higher is better)

TCP backpressure breakage

Can we achieve the same for Pods?

* 8264 MTU for data page alignment in GRO

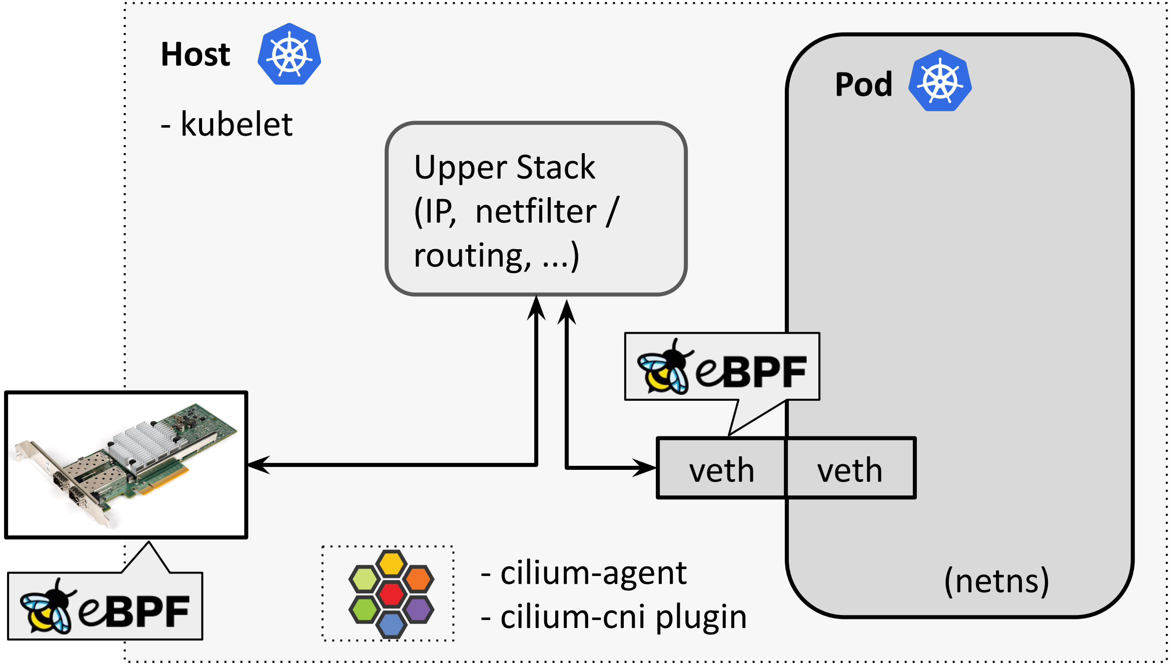
Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO off, 8264 MTU

Receiver: taskset -a -c <core> tcp_mmap -s (non-zerocopy mode), Sender: taskset -a -c <core> tcp_mmap -H <dst host>

Cilium Datapath Architecture:

Reducing netns Overhead:

- BPF Host Routing

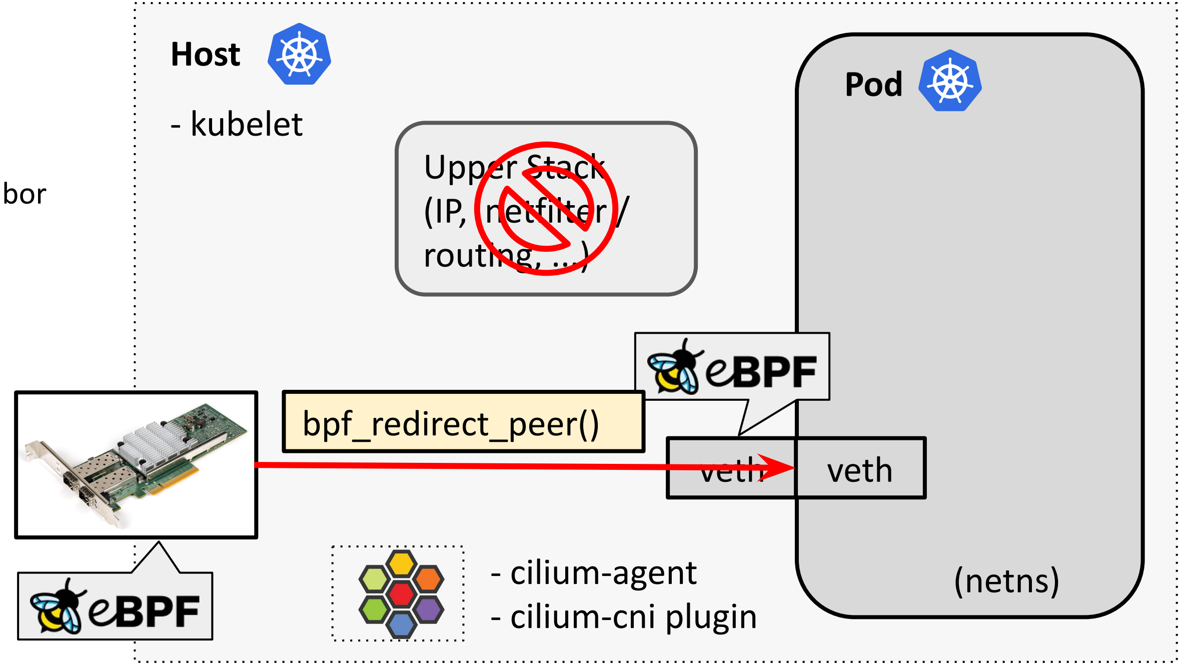


Kernel 5.10: bpf_redirect_peer and bpf_redirect_neigh helpers (Daniel Borkmann)

Cilium Datapath Architecture:

Reducing netns Overhead:

- BPF Host Routing
  - ↪ Routing only via tc BPF layer
  - ↪ Fast netns switch on ingress
  - ↪ Helper for fib + dynamic neighbor resolution on egress

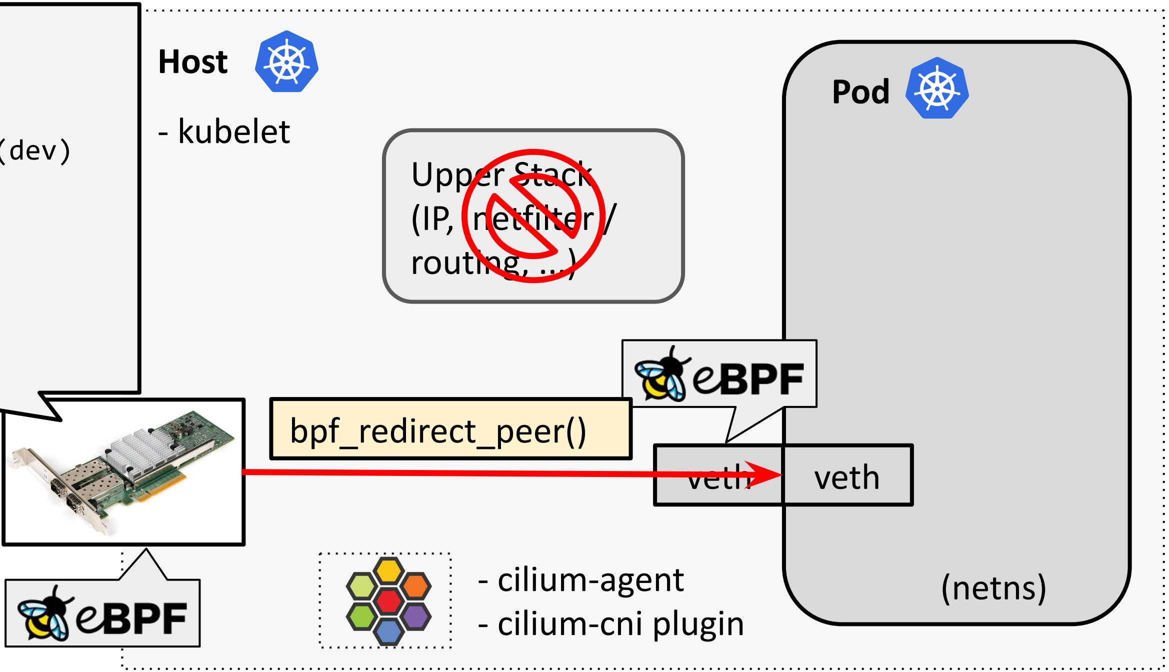


Kernel 5.10: bpf_redirect_peer and bpf_redirect_neigh helpers (Daniel Borkmann)

Cilium Datapath Architecture:

Redirection Internals:

```
dev = ops->ndo_get_peer_dev(dev)
skb_scrub_packet()
skb->dev = dev
sch_handle_ingress():
 - goto another_round
 - no per-CPU backlog queue
```

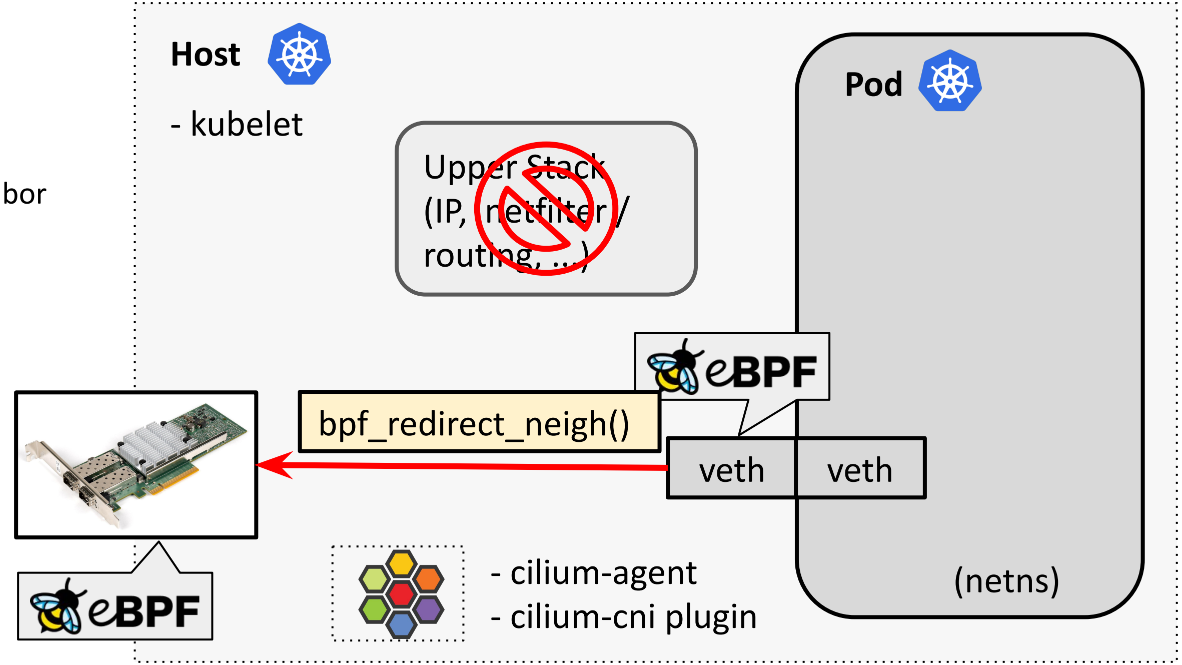


Kernel 5.10: bpf_redirect_peer and bpf_redirect_neigh helpers (Daniel Borkmann)

Cilium Datapath Architecture:

Reducing netns Overhead:

- BPF Host Routing
  - ↪ Routing only via tc BPF layer
  - ↪ Fast netns switch on ingress
  - ↪ Helper for fib + dynamic neighbor resolution on egress

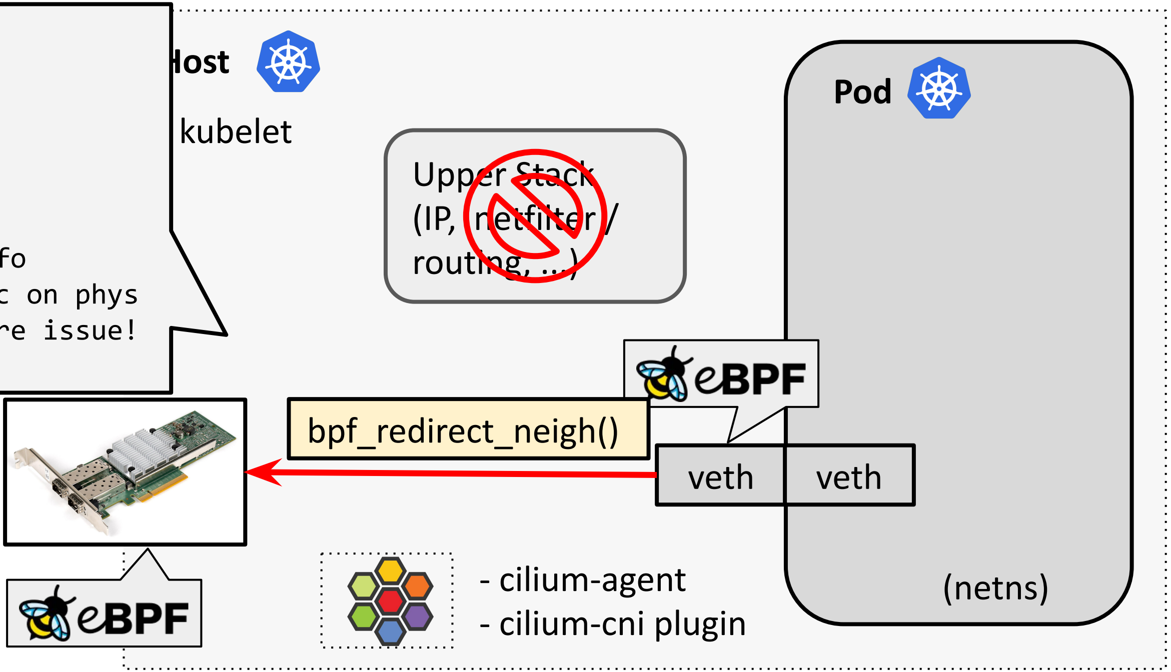


Kernel 5.10: bpf_redirect_peer and bpf_redirect_neigh helpers (Daniel Borkmann)

Cilium Datapath Architecture:

Redirection Internals:

```
ip_route_output_flow()
skb_dst_set()
ip_finish_output2()
- fills in neighbor (L2) info
- retains skb->sk till Qdisc on phys
-> fixes the TCP backpressure issue!
```

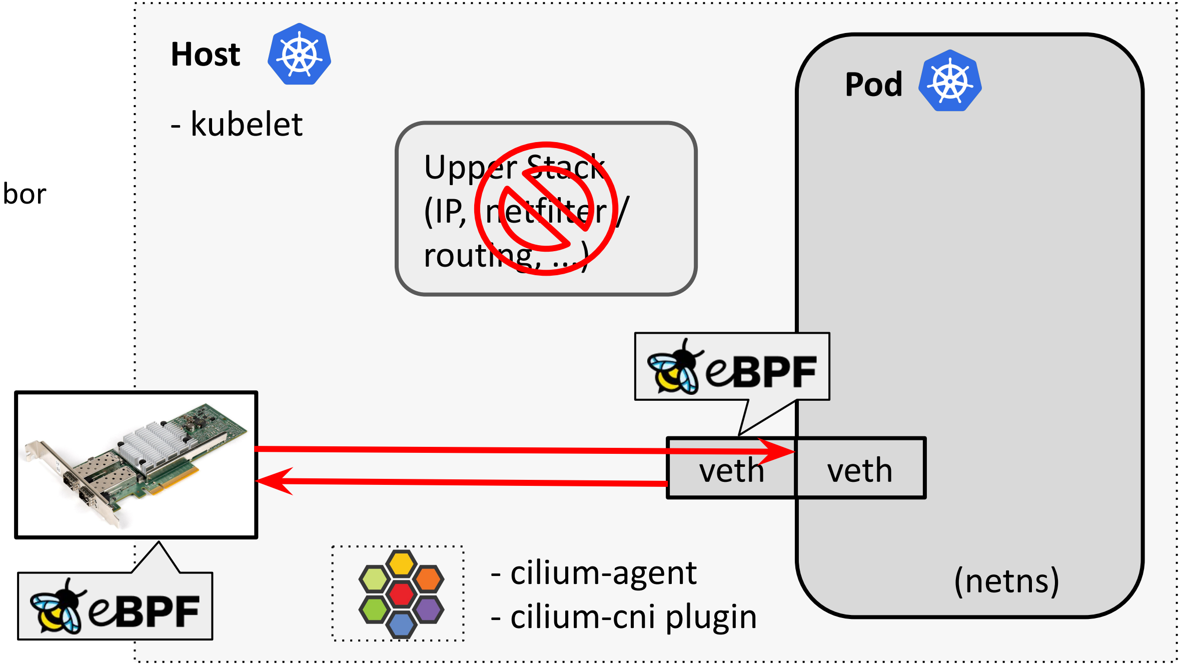


Kernel 5.10: bpf_redirect_peer and bpf_redirect_neigh helpers (Daniel Borkmann)

Cilium Datapath Architecture:

Reducing netns Overhead:

- BPF Host Routing
  - ↪ Routing only via tc BPF layer
  - ↪ Fast netns switch on ingress
  - ↪ Helper for fib + dynamic neighbor resolution on egress

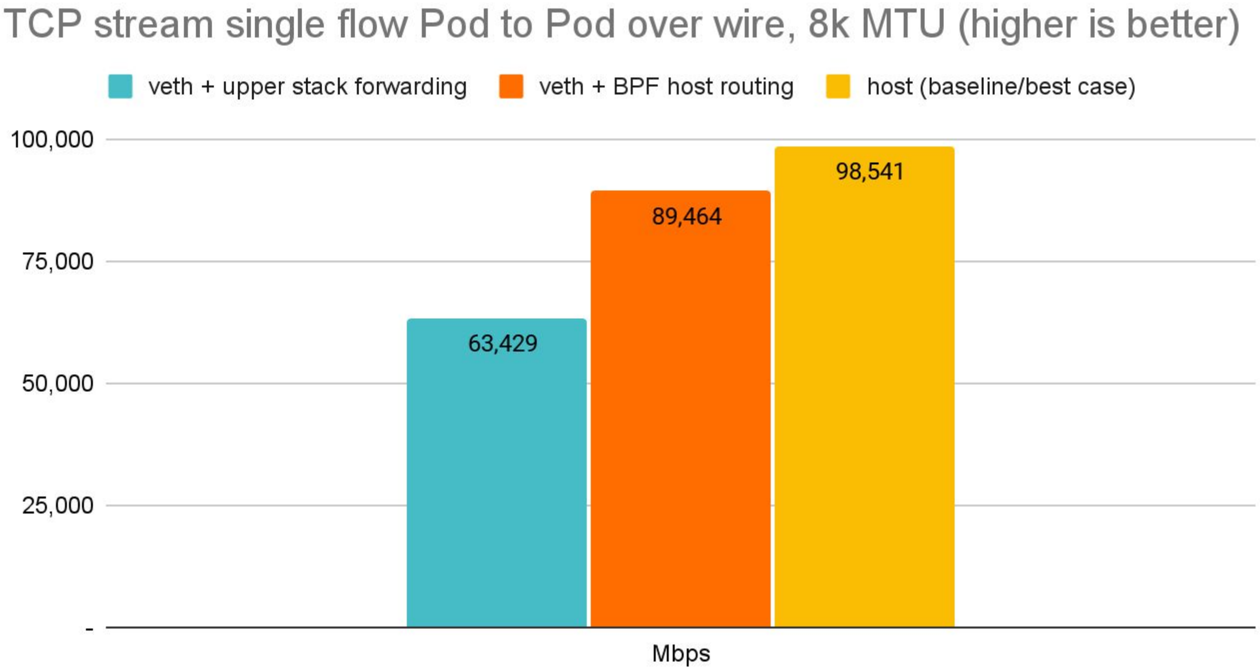


Kernel 5.10: bpf_redirect_peer and bpf_redirect_neigh helpers (Daniel Borkmann)

Cilium Datapath Architecture:

Reducing netns Overhead:

- BPF Host Routing



* 8264 MTU for data page alignment in GRO

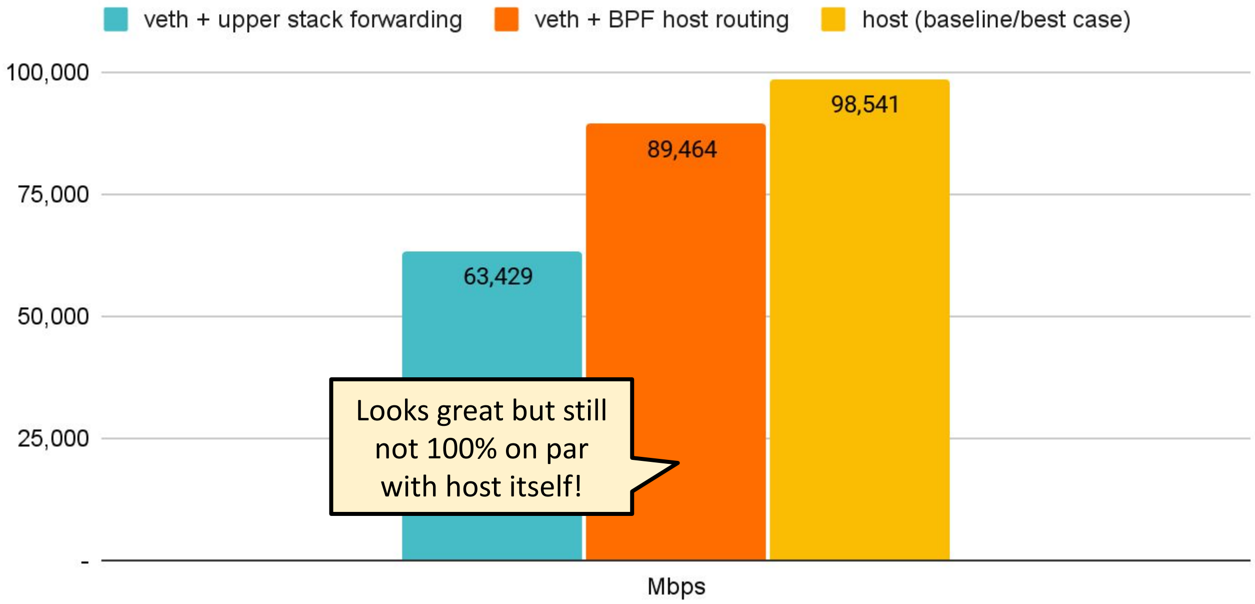
Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO off, 8264 MTU
Receiver: taskset -a -c <core> tcp_mmap -s (non-zerocopy mode), Sender: taskset -a -c <core> tcp_mmap -H <dst host>

Cilium Datapath Architecture:

Reducing netns Overhead:

- BPF Host Routing

TCP stream single flow Pod to Pod over wire, 8k MTU (higher is better)



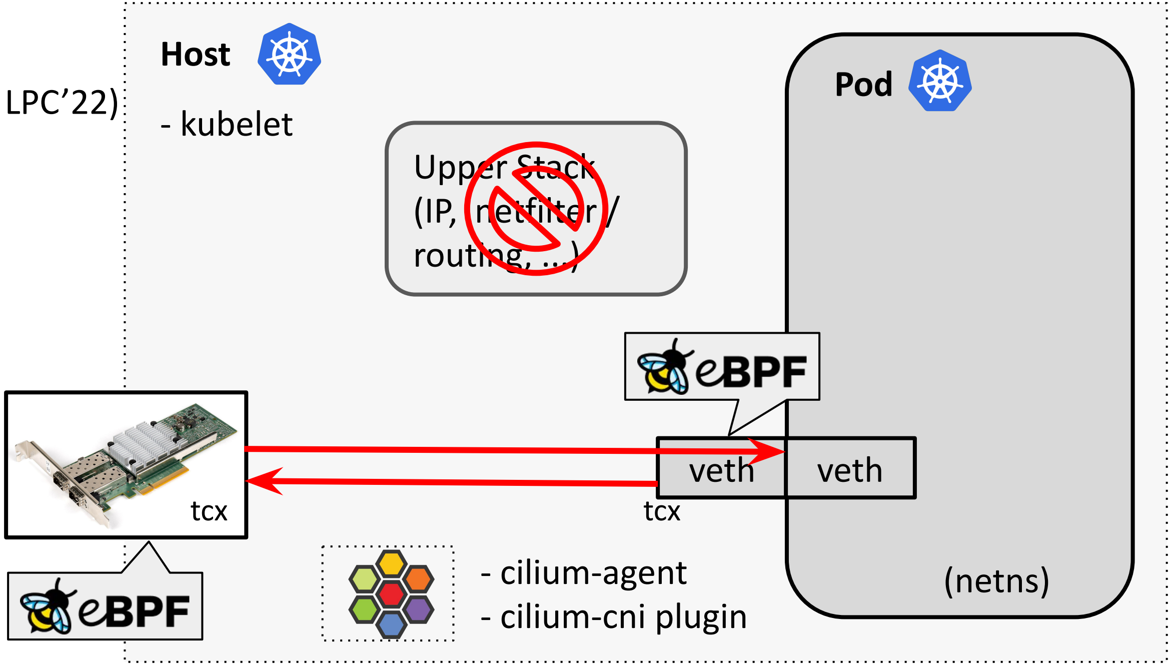
Looks great but still not 100% on par with host itself!

* 8264 MTU for data page alignment in GRO

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO off, 8264 MTU
Receiver: taskset -a -c <core> tcp_mmap -s (non-zerocopy mode), Sender: taskset -a -c <core> tcp_mmap -H <dst host>

Cilium Datapath Architecture:

Reducing netns Overhead:
- BPF Host Routing
- tcx-based BPF datapath (see LPC’22)

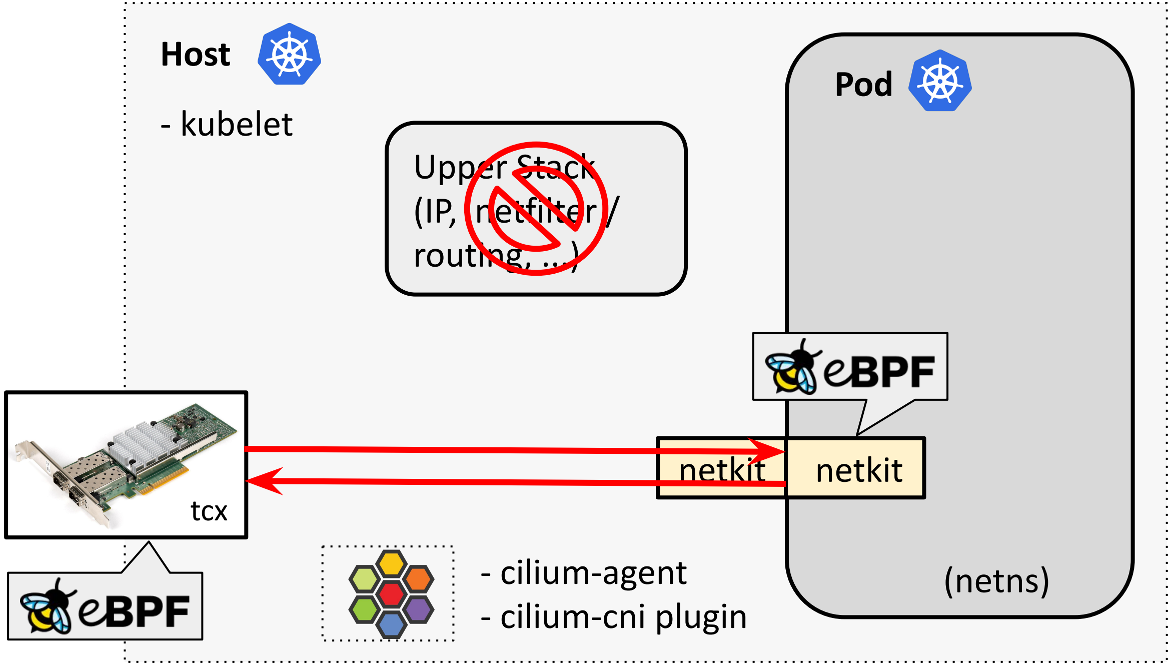


Kernel 6.6: tcx infrastructure and BPF link support (Daniel Borkmann)

Cilium Datapath Architecture:

Reducing netns Overhead:

- BPF Host Routing
- tcx-based BPF datapath
- netkit devices for Pods

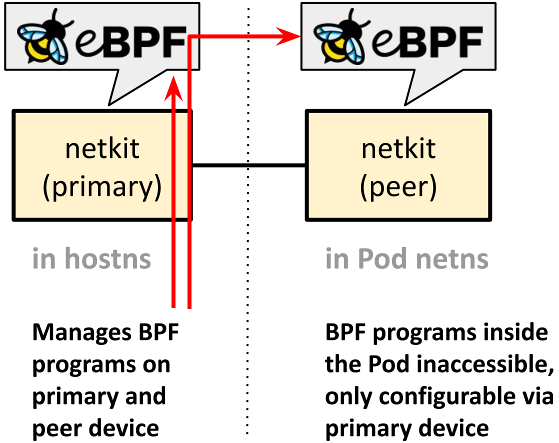


Kernel 6.7: netkit devices (Daniel Borkmann, Nikolay Aleksandrov), **LWN coverage:** The BPF-programmable network device

Deep Dive: Cilium and netkit for Pods

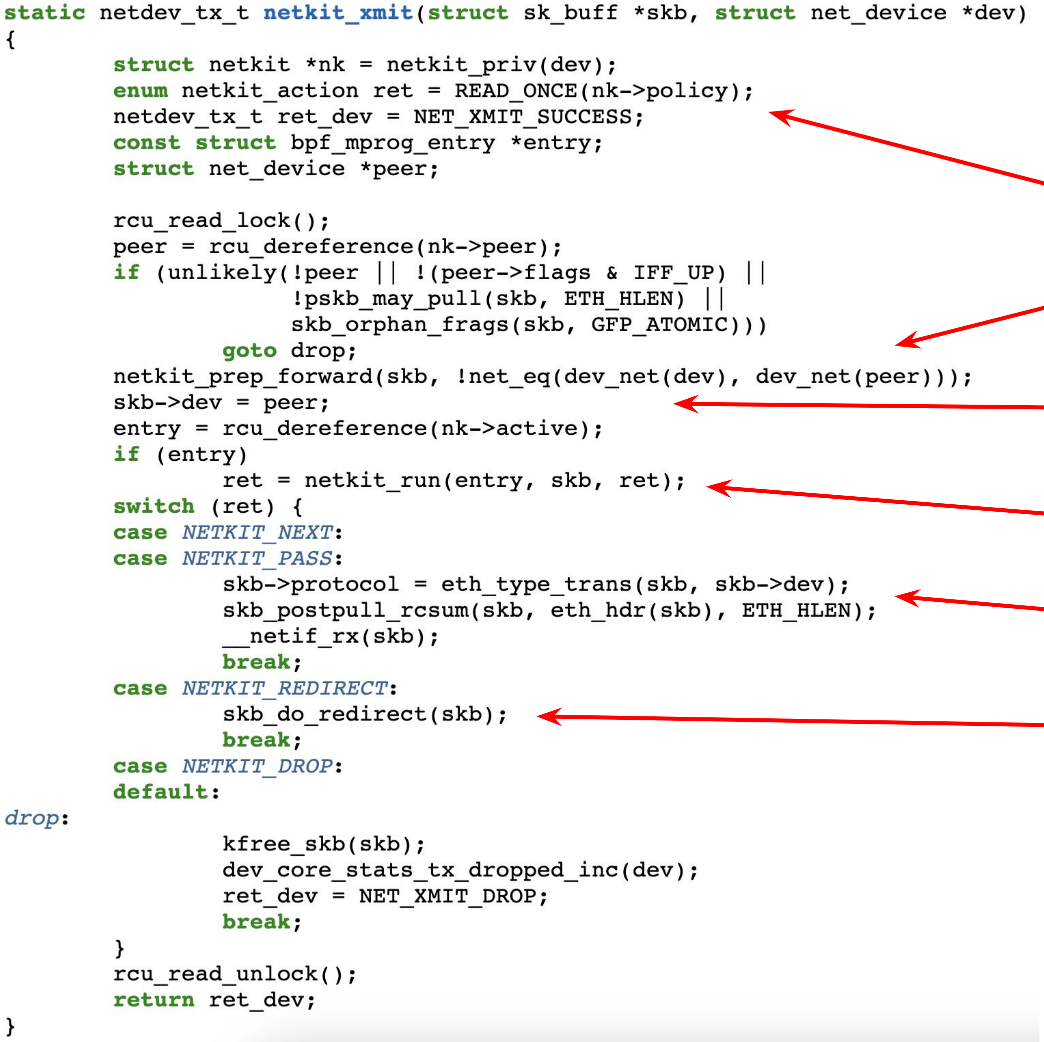
netkit programmable virtual devices for BPF:

- Going forward, Cilium's CNI code will set up netkit devices for Pods instead of veth
- BPF program via bpf_mprog is part of the driver's xmit routine, allowing fast egress netns switch
- Driver implements ndo_get_peer_dev, allowing fast ingress netns switch
- Configurable as L3 device (default) or L2 device
- Configurable default drop-all if no BPF is attached



Manages BPF programs on primary and peer device

BPF programs inside the Pod inaccessible, only configurable via primary device



```
static netdev_tx_t netkit_xmit(struct sk_buff *skb, struct net_device *dev)
{
        struct netkit *nk = netkit_priv(dev);
        enum netkit_action ret = READ_ONCE(nk->policy);
        netdev_tx_t ret_dev = NET_XMIT_SUCCESS;
        const struct bpf_mprog_entry *entry;
        struct net_device *peer;

        rcu_read_lock();
        peer = rcu_dereference(nk->peer);
        if (unlikely(!peer || !(peer->flags & IFF_UP) ||
                     !pskb_may_pull(skb, ETH_HLEN) ||
                     skb_orphan_frags(skb, GFP_ATOMIC)))
                goto drop;
        netkit_prep_forward(skb, !net_eq(dev_net(dev), dev_net(peer)));
        skb->dev = peer;
        entry = rcu_dereference(nk->active);
        if (entry)
                ret = netkit_run(entry, skb, ret);
        switch (ret) {
        case NETKIT_NEXT:
        case NETKIT_PASS:
                skb->protocol = eth_type_trans(skb, skb->dev);
                skb_postpull_rcsum(skb, eth_hdr(skb), ETH_HLEN);
                __netif_rx(skb);
                break;
        case NETKIT_REDIRECT:
                skb_do_redirect(skb);
                break;
        case NETKIT_DROP:
        default:
drop:
                kfree_skb(skb);
                dev_core_stats_tx_dropped_inc(dev);
                ret_dev = NET_XMIT_DROP;
                break;
        }
        rcu_read_unlock();
        return ret_dev;
}
```

default policy if nothing attached

skb scrubbing

netns switch, provides hostns context for bpf_fib_lookup(), bpf_redirect*(), etc.

executes active bpf_mprog array

for redirect into local hostns

performs direct redirect to phys device

Deep Dive: netkit Internals /2

BPF entry point

```
static __always_inline int
netkit_run(const struct bpf_mprog_entry *entry, struct sk_buff *skb,
           enum netkit_action ret)
{
        const struct bpf_mprog_fp *fp;
        const struct bpf_prog *prog;

        bpf_mprog_foreach_prog(entry, fp, prog) {
                bpf_compute_data_pointers(skb);
                ret = bpf_prog_run(prog, skb);
                if (ret != NETKIT_NEXT)
                        break;
        }
        return ret;
}
```

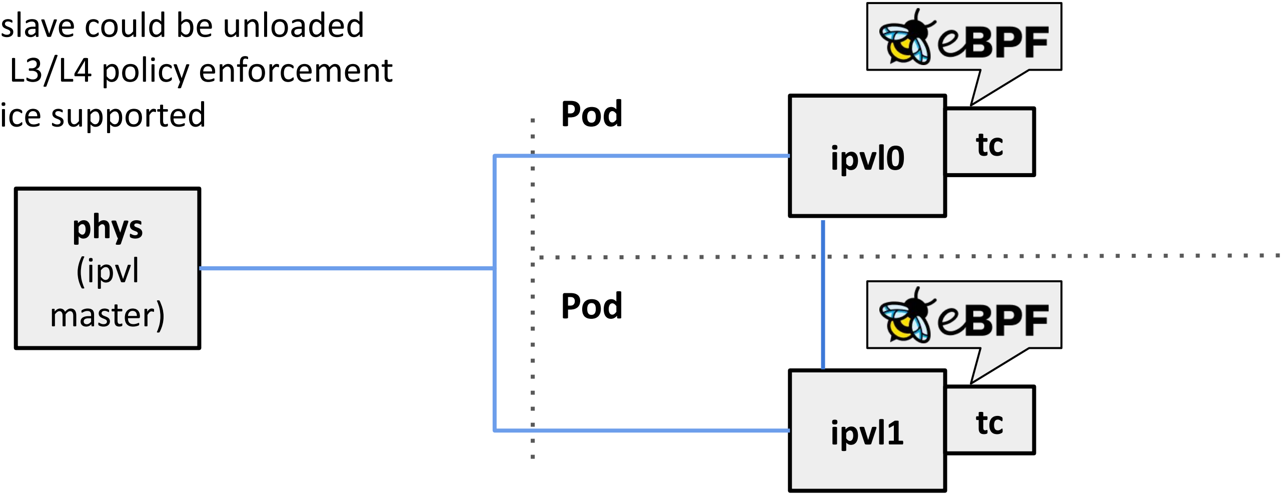
```
struct bpf_mprog_fp {
        struct bpf_prog *prog;
};

struct bpf_mprog_cp {
        struct bpf_link *link;
};

struct bpf_mprog_entry {
        struct bpf_mprog_fp fp_items[BPF_MPROG_MAX];
        struct bpf_mprog_bundle *parent;
};

struct bpf_mprog_bundle {
        struct bpf_mprog_entry a;
        struct bpf_mprog_entry b;
        struct bpf_mprog_cp cp_items[BPF_MPROG_MAX];
        struct bpf_prog *ref;
        atomic64_t revision;
        u32 count;
};
```

a/b swappable mprog bundle

Cilium and ipvlan

Cilium used to have limited ipvlan support in the past

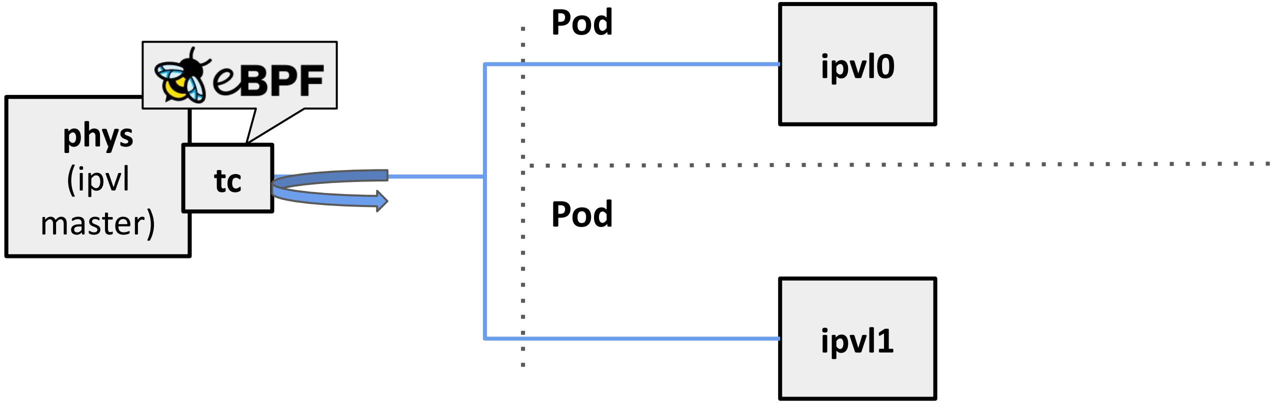
- Cilium CNI sets up ipvlan slave device for the target netns with a simple BPF program doing a tailcall
  - BPF tailcall map resides in hostns so that Cilium agent has access to it (it cannot enter Pod's netns)
- Cilium's bpf_lxc program added to tailcall map
- Limitations:
  - L3 mode comes with netfilter asymmetry -> L3S needed
  - BPF programs @ ipvlan slave could be unloaded
  - No L7 proxy (host), only L3/L4 policy enforcement
  - Only single physical device supported



Cilium and ipvlan

Cilium used to have limited ipvlan support in the past

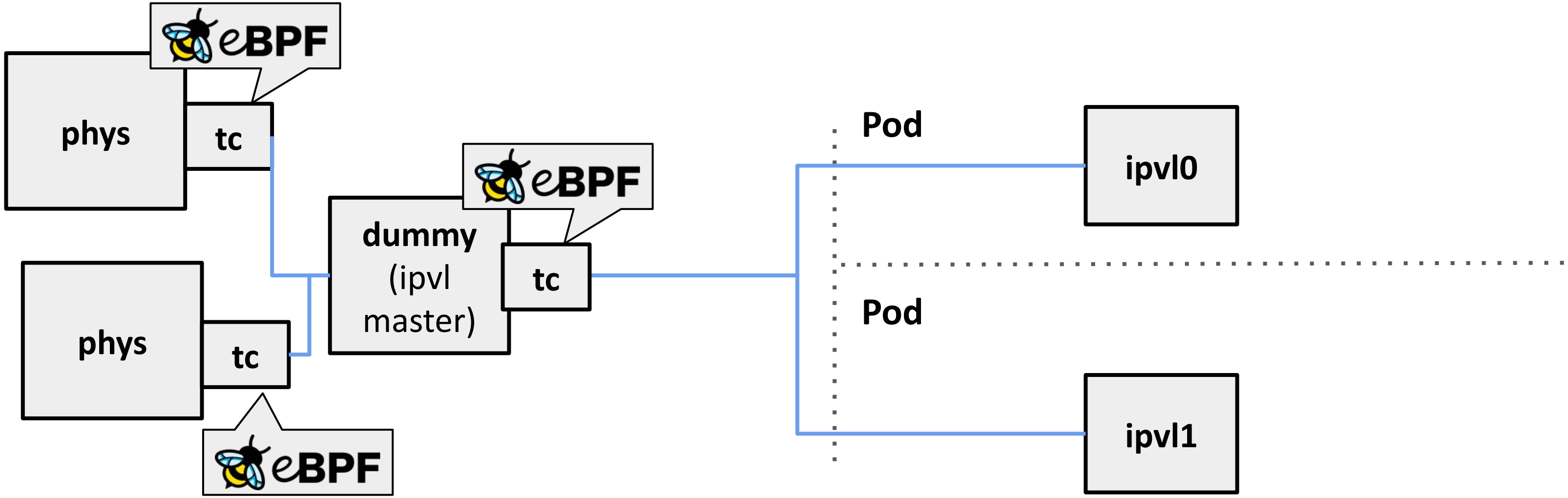
- What could have been done better? Cilium could have used L3(S) private mode (forces all traffic out to hostns)
- Cilium's bpf_lxc program added to tailcall map which is accessed at tc egress of physical device
  - Pod to Pod communication goes to phys egress, bpf_redirect to ingress to loop back to Pod
- Limitations (remainder):
  - Traffic classification for tailcall map entry needed at tc egress, might have allowed Pod to spoof their source
  - L3 mode comes with netfilter asymmetry -> L3S needed
  - Only single physical device supported



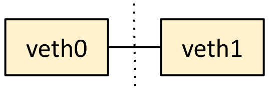
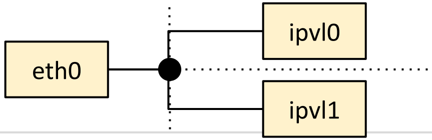
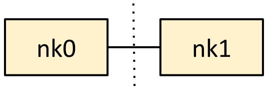
Cilium and ipvlan

Cilium used to have limited ipvlan support in the past

- How might multiple physical devices look in this architecture picture?

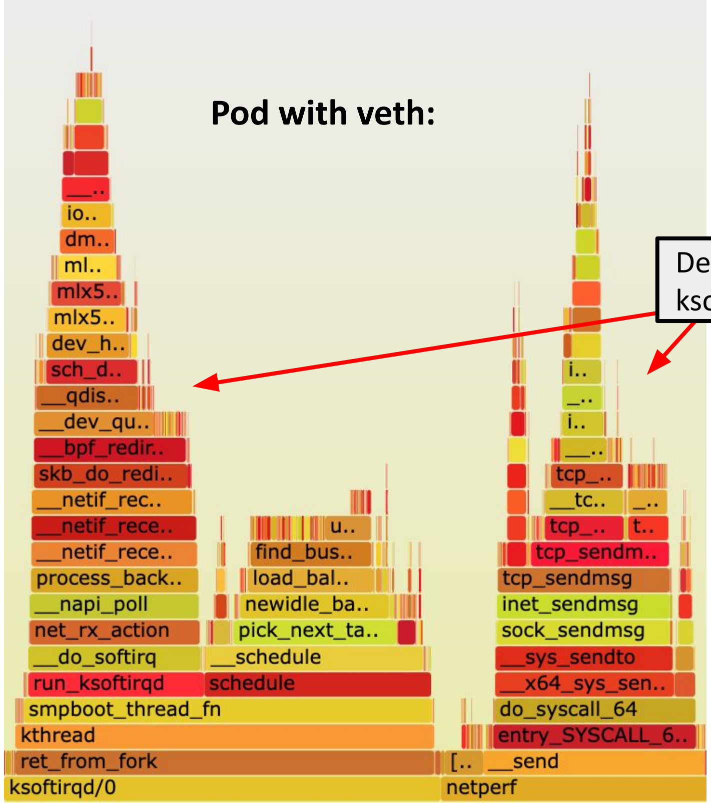


Comparison veth, ipvlan, netkit

	veth	ipvlan	netkit
Operation mode:	L2	L3 (or L2)	L3 (or L2)
Device "legs":	pair (e.g. 1 host, 1 Pod)  veth0 veth1	1 "master" device (e.g. physical device), n "slave" devices  eth0 ipvl0 ipvl1	pair (e.g. 1 host, 1 Pod) with "primary" and "peer" device  nk0 nk1
BPF programming:	tc(x) BPF on host device*	In host with tc(x) via "master" device (only entity in host) *	In Pod, BPF is native part of "peer" device internals
Routing:	L2 gateway (+ kernel FIB)	ipvlan internal FIB + kernel FIB	kernel FIB e.g. bpf_fib_lookup()
Problems:	Needs L2 neigh resolution, Higher overhead due to per-CPU backlog queue, native XDP support but very slow and hard to use .	Inflexible for multiple physical devices & troubleshooting, cumbersome to program BPF on "master". ipvlan needs to be operated in L3/private mode for Pod policy enforcement.	Still one device per Pod inside host, for some use-cases the host device can be removed fully (wip).

(* It needs to be inside host so that BPF programs cannot be detached from app inside Pod)

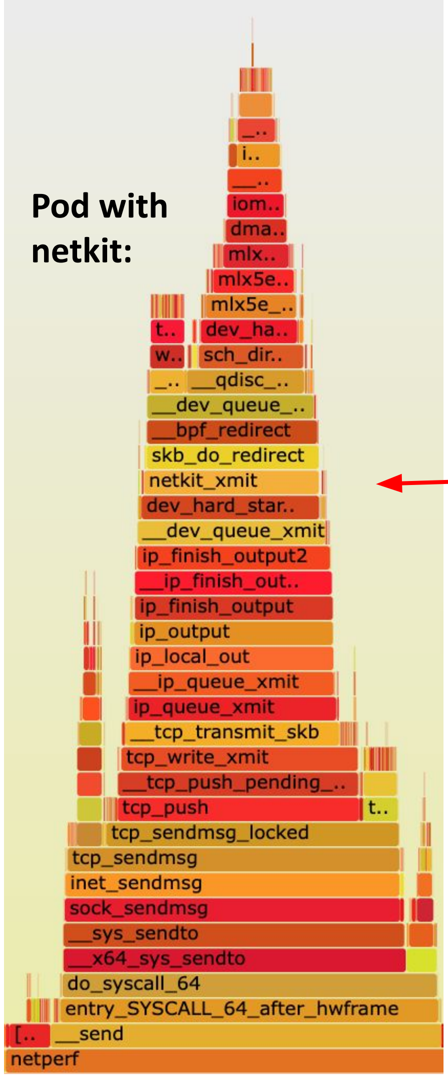
veth vs netkit: Backlog Queue



Pod with veth:

Deferral to ksoftirqd

Pod with netkit:

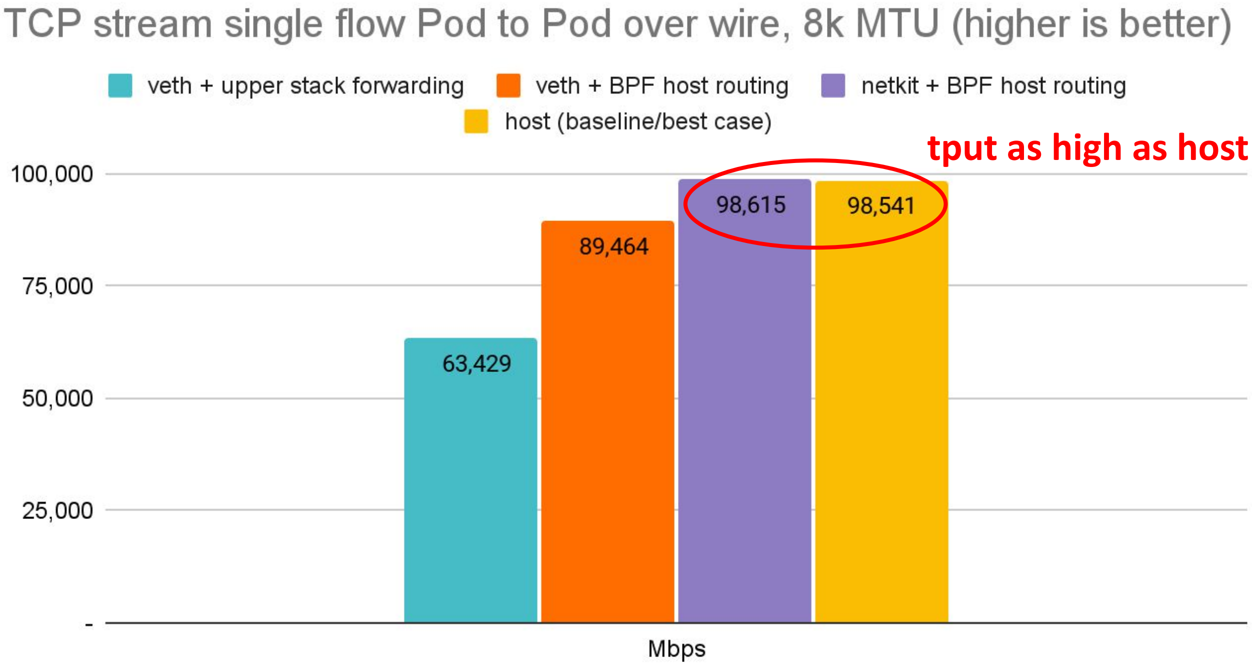


Remains in process context all the way, allows for better scheduler accounting

Cilium Datapath Architecture:

Reducing netns Overhead:

- BPF Host Routing
- tcx-based BPF datapath layer
- netkit devices for Pods



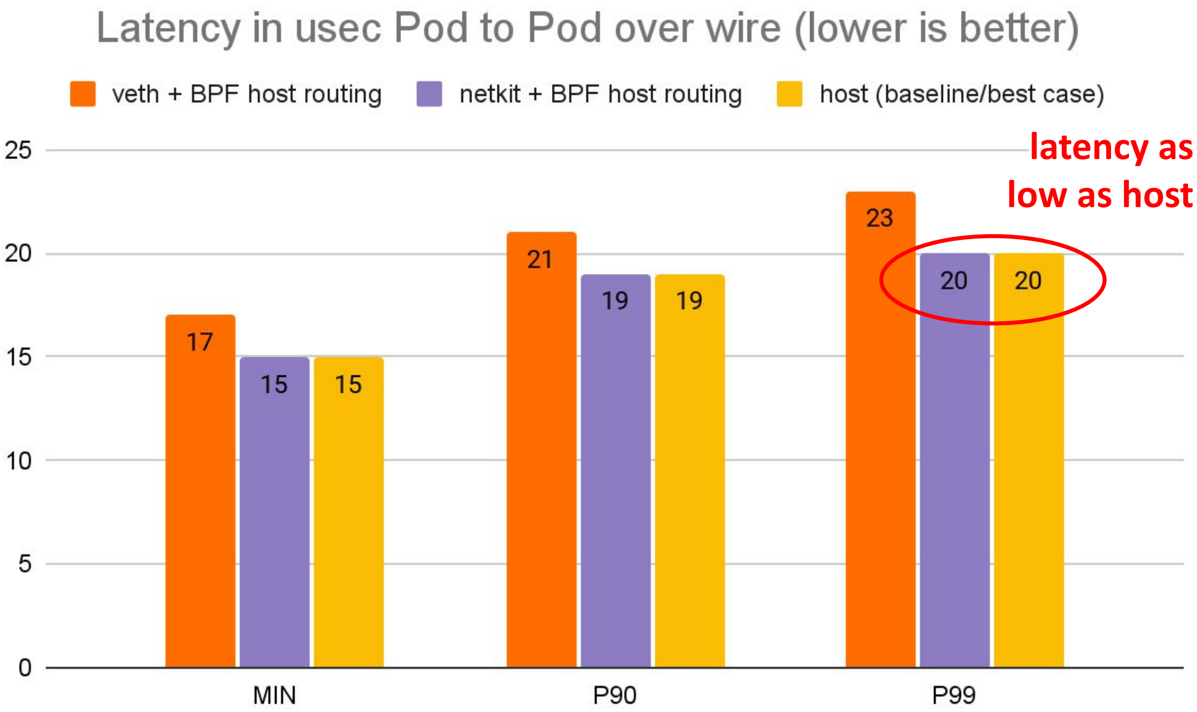
* 8264 MTU for data page alignment in GRO

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO off, 8264 MTU
Receiver: taskset -a -c <core> tcp_mmap -s (non-zerocopy mode), Sender: taskset -a -c <core> tcp_mmap -H <dst host>

Cilium Datapath Architecture:

Reducing netns Overhead:

- BPF Host Routing
- tcx-based BPF datapath layer
- netkit devices for Pods



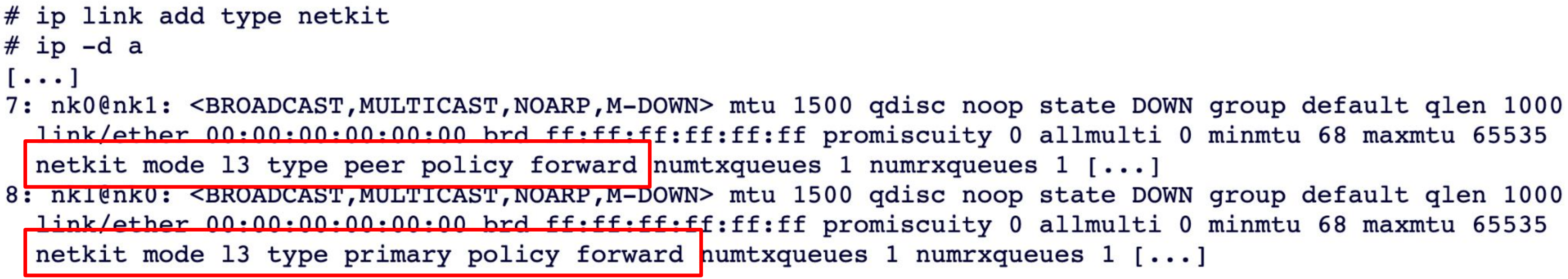
latency as low as host

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO off
netperf -t TCP_RR -H <remote pod> -- -O MIN_LATENCY,P90_LATENCY,P99_LATENCY,THROUGHPUT

netkit: Ongoing work

iproute2 and vishvananda/netlink library support

- Goal: Basic device setup and introspection support



```
# ip link add type netkit
# ip -d a
[...]
7: nk0@nk1: <BROADCAST,MULTICAST,NOARP,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
  link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff promiscuity 0 allmulti 0 minmtu 68 maxmtu 65535
  netkit mode l3 type peer policy forward numtxqueues 1 numrxqueues 1 [...]
8: nk1@nk0: <BROADCAST,MULTICAST,NOARP,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
  link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff promiscuity 0 allmulti 0 minmtu 68 maxmtu 65535
  netkit mode l3 type primary policy forward numtxqueues 1 numrxqueues 1 [...]
```

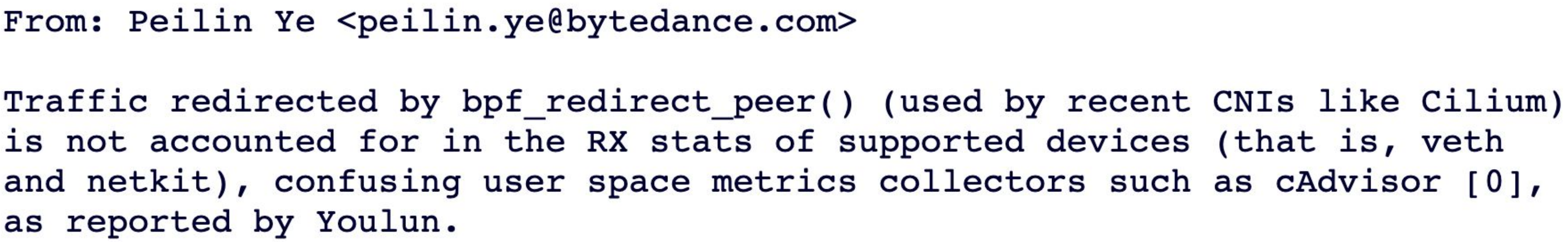
- Support base setup and delegate BPF program management to applications (via libbpf, ebpf-go)
- vishvananda/netlink needed for Cilium CNI plugin integration (Go-based)

Patches: ip, link: Add support for netkit (Daniel Borkmann)

netkit: Ongoing work

Fixing networking stats for netkit in general and for peer-redirection

- Goal: Proper network stats accounting for netkit and veth



```
From: Peilin Ye <peilin.ye@bytedance.com>

Traffic redirected by bpf_redirect_peer() (used by recent CNIs like Cilium)
is not accounted for in the RX stats of supported devices (that is, veth
and netkit), confusing user space metrics collectors such as cAdvisor [0],
as reported by Youlun.
```

- Fix is calling dev_sw_netstats_rx_add() in skb_do_redirect() and move netkit & veth to dev->tstats
- Guard if drivers implementing ndo_get_peer_dev and do not use dev->tstats
- Suggestion from Jakub Kicinski to move {l,t,d}stats allocation into net core

Patches: bpf_redirect_peer fixes, v3 (Daniel Borkmann, Jakub Kicinski, Nikolay Aleksandrov et al.)

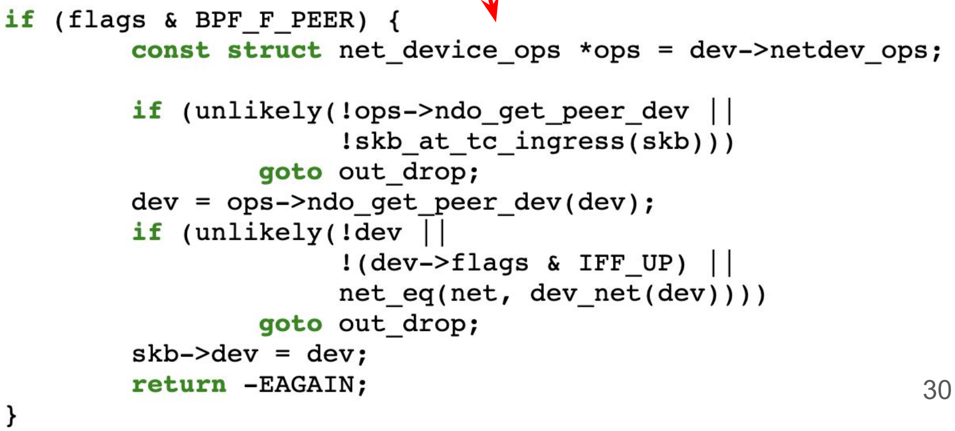
netkit: Ongoing work

Adding peer pointer into struct net_device

- Goal: Get rid of ndo_get_peer_dev entirely and add peer pointer to net_device (suggestion from Jakub Kicinski)

- The latter is only implemented by veth and netkit
- Helps performance for ingress direction due to the current indirect call in skb_do_redirect()

- Short-term: INDIRECT_CALL_1() macro can be utilized on the ndo if driver is built-in
- Mid-term: Rework veth and netkit and remove private peer pointers in favor of dev->peer



```
if (flags & BPF_F_PEER) {
        const struct net_device_ops *ops = dev->netdev_ops;

        if (unlikely(!ops->ndo_get_peer_dev ||
                     !skb_at_tc_ingress(skb)))
                goto out_drop;
        dev = ops->ndo_get_peer_dev(dev);
        if (unlikely(!dev ||
                     !(dev->flags & IFF_UP) ||
                     net_eq(net, dev_net(dev))))
                goto out_drop;
        skb->dev = dev;
        return -EAGAIN;
}
```

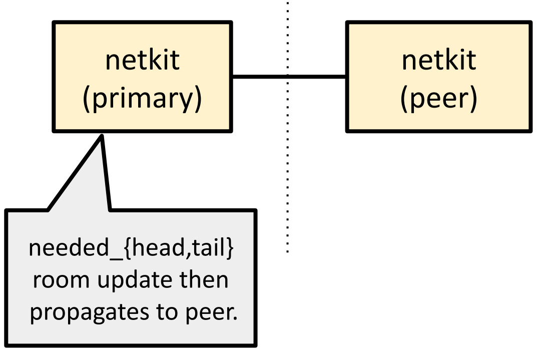
netkit: Ongoing work

Experimenting with head/tailroom customization

- Goal: Being able to control dev->needed_headroom and dev->needed_tailroom

- Could benefit datapath performance under tunneling (vxlan, geneve) or encryption (wireguard)
    - Potentially avoids pskb_expand_head() reallocation costs

- Idea: Have actual IFLA_HEADROOM and IFLA_TAILROOM attributes to dump and set on a device
- needed_{head,tail}room is by default 0, vxlan/geneve adjusts needed_headroom, wireguard also needed_tailroom
- Performance benefit: to be measured, references from old patches mention costs around 5% on realloc



needed_{head,tail} room update then propagates to peer.

WIP: head/tailroom getter/setter in rtnetlink

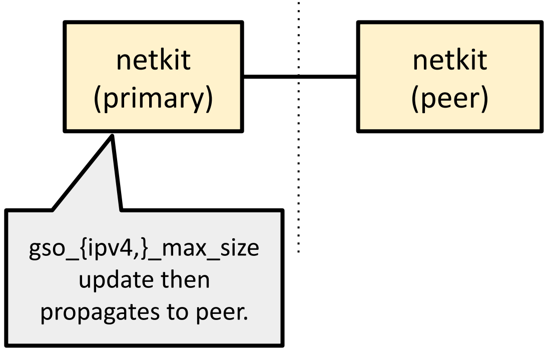
netkit: Ongoing work

Adding new ndo for setting dev->gso_{ipv4,}_max_size

- Goal: Enabling BIG TCP for Pods without having to restart Pods

- Cilium agent is not able to exec into the Pod's netns at runtime and mounting host procfs into Cilium container is not desired (security reasons). Only the Cilium CNI plugin has access when setting up devices.

- Downside: Enabling BIG TCP on an existing cluster requires restart of application Pod
- New ndo for updating dev->gso_{ipv4,}_max_size in similar style as dev->needed_{head,tail}room would be desirable.. e.g. picks max of primary/peer and applies it to both

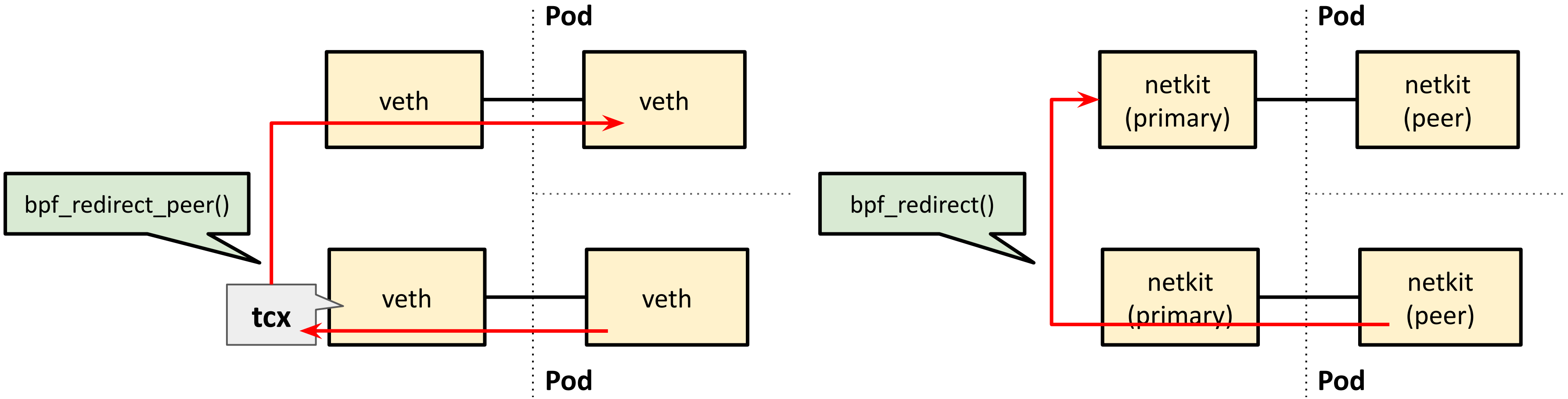


netkit: Ongoing work

veth vs netkit difference in terms of bpf_redirect_peer

- Goal: Usability improvements to avoid users running into pitfalls

- Performance-wise local node's Pod-Pod redirection optimizations are on-par for veth and netkit
- Just that for veth bpf_redirect_peer (ingress->ingress) is needed whereas netkit bpf_redirect (egress->egress)
- Perhaps rather documentation/awareness issue?



netkit: Future work

Semi-related to netkit: Removing indirect calls for IPv6 in bpf_fib_lookup()

- Goal: Further improve bpf_fib_lookup() IPv6 performance by removing indirect calls from fast-path.
  Useful when netkit is utilising the latter out of its BPF programs.

- Worst case 4 indirect calls from a single BPF helper call (!) via bpf_ipv6_fib_lookup():
  - ipv6_stub->fib6_{table_,}lookup()
  - ipv6_stub->fib6_select_path()
  - ipv6_stub->ip6_mtu_from_fib6()
  - ipv6_bpf_stub->ipv6_dev_get_saddr()

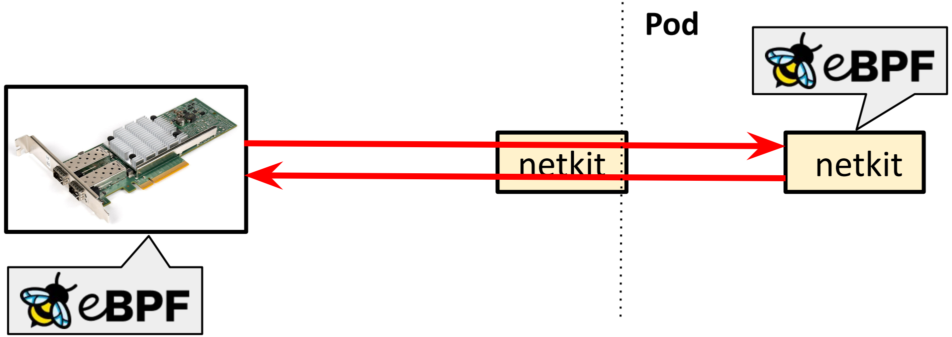
- Two options to overcome:
  - Remove IPv6 tristate from Kbuild, move to bool and then successively get rid of the stub helpers (my preference)
  - INDIRECT_CALL_1() wrappers for the case when IPv6 is built-in

netkit: Future work

BPF kfunc for dev_queue_xmit_nit to allow for tcpdump at specific points in BPF program

- Goal: Improve troubleshooting for Pod-traffic

- bpf_redirect_peer() skips host device on ingressing traffic (netkit, veth), and bpf_redirect() from the BPF program inside the netkit device skips host device on egressing traffic
- One either needs to nsenter into Pod for troubleshooting or capture all traffic on the phys device
- BPF kfunc for dev_queue_xmit_nit-like functionality helps to dynamically insert tcpdump-tracing at custom points

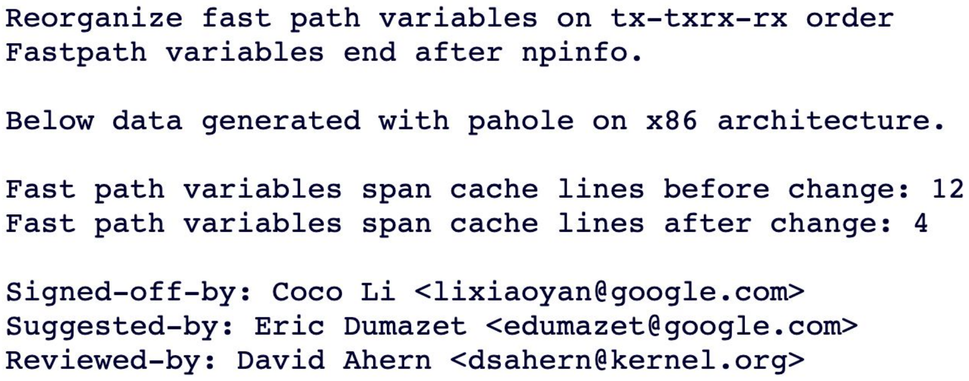


netkit: Future work

Semi-related to netkit: Reorganize ndos in struct net_device_ops

- Goal: Reduce cacheline access for dev->netdev_ops to improve performance

- Picking up on the work Coco Li kicked off in order to look at fast-path RX ndo ops and fast-path TX ndo ops
- Wrapping them into __cacheline_group_{begin,end} markers with member and size assertions
- struct net_device had nice gains:



```
Reorganize fast path variables on tx-txrx-rx order
Fastpath variables end after npinfo.

Below data generated with pahole on x86 architecture.

Fast path variables span cache lines before change: 12
Fast path variables span cache lines after change: 4

Signed-off-by: Coco Li <lixiaoyan@google.com>
Suggested-by: Eric Dumazet <edumazet@google.com>
Reviewed-by: David Ahern <dsahern@kernel.org>
```

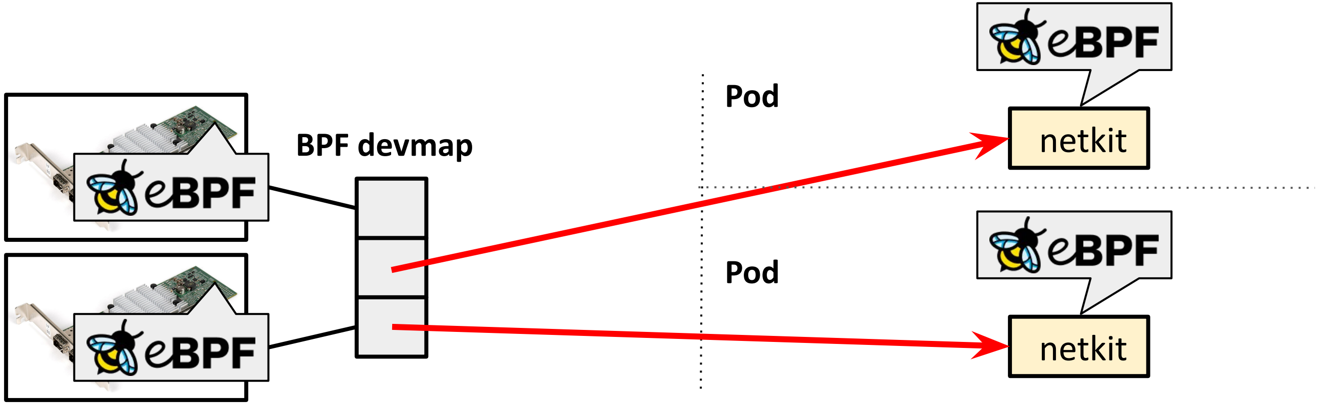
Patches: optimize cacheline access (Coco Li)

netkit: Future work

netkit single device mode

- Goal: Could we even go further and remove the host-facing device?

- Single netkit device mode could act as sink within Pod
- BPF devmap in hostns keeps the device pointers of the peer device as map values, addressable by id key.
  Modified bpf_redirect_peer() would look up BPF devmap instead.
- BPF devmap can be shared among multiple phys devices inside hostns

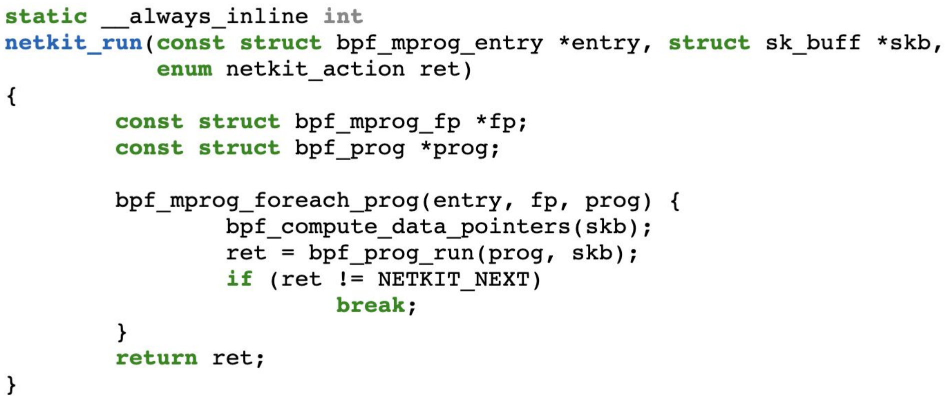


netkit: Future work

Getting rid of indirect call overhead for bpf_mprog

- Goal: Get rid of indirect calls when invoking BPF programs (tcx, netkit)

- Similar to XDP use dynamically generated branch funnel
- More involved due to the array as opposed to just single program invocation, but might be needed anyway for XDP bpf_mprog support
- Needs benchmarking as branch funnel for netkit will be much larger than XDP



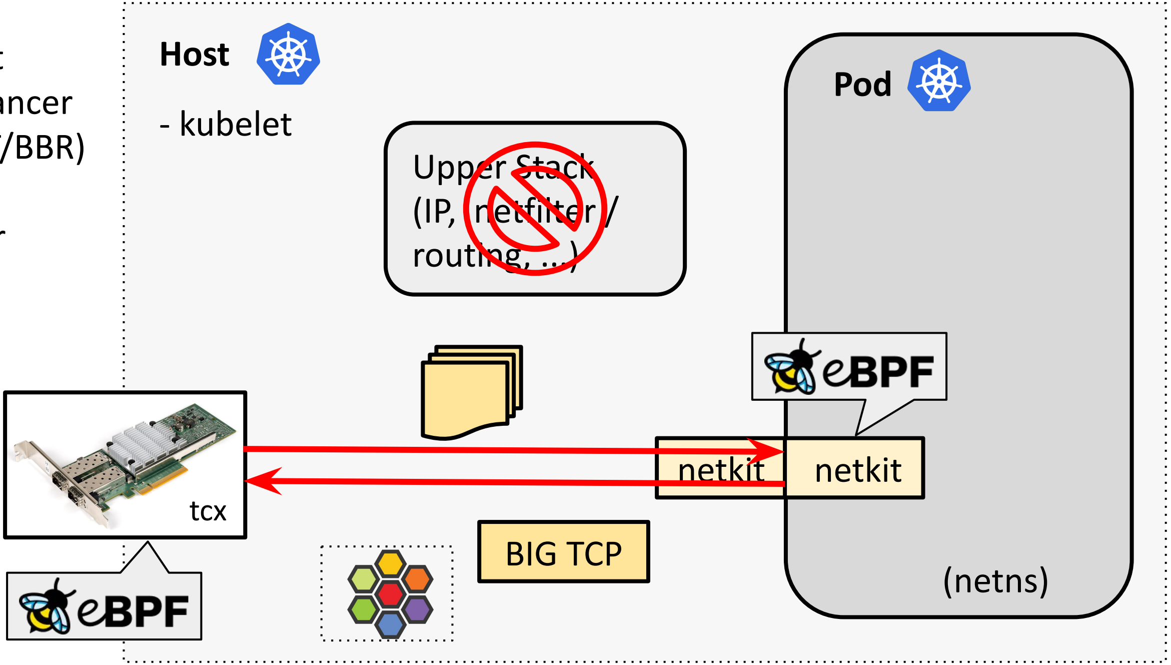
```
static __always_inline int
netkit_run(const struct bpf_mprog_entry *entry, struct sk_buff *skb,
           enum netkit_action ret)
{
        const struct bpf_mprog_fp *fp;
        const struct bpf_prog *prog;

        bpf_mprog_foreach_prog(entry, fp, prog) {
                bpf_compute_data_pointers(skb);
                ret = bpf_prog_run(prog, skb);
                if (ret != NETKIT_NEXT)
                        break;
        }
        return ret;
}
```

Cilium Datapath Architecture (journey 2019 - today):

All Building Blocks:
- BPF kube-proxy replacement
- XDP-based Service Load-Balancer
- Bandwidth Manager (fq/EDT/BBR)
- BPF Host Routing
- tcx-based BPF datapath layer
- netkit devices for Pods

Pushing even further:
- BIG TCP (IPv4/IPv6)

Future integration:
- TCP usec resolution (v6.7)
- BBRv3 (once upstream)



ISOVALENT

Thank you! Questions?

github.com/cilium/cilium

cilium.io

ebpf.io

BPF Host Routing

tcx BPF datapath

netkit devices

BIG TCP for IPv4/IPv6