meta netdevices

Daniel Borkmann, Isovalent

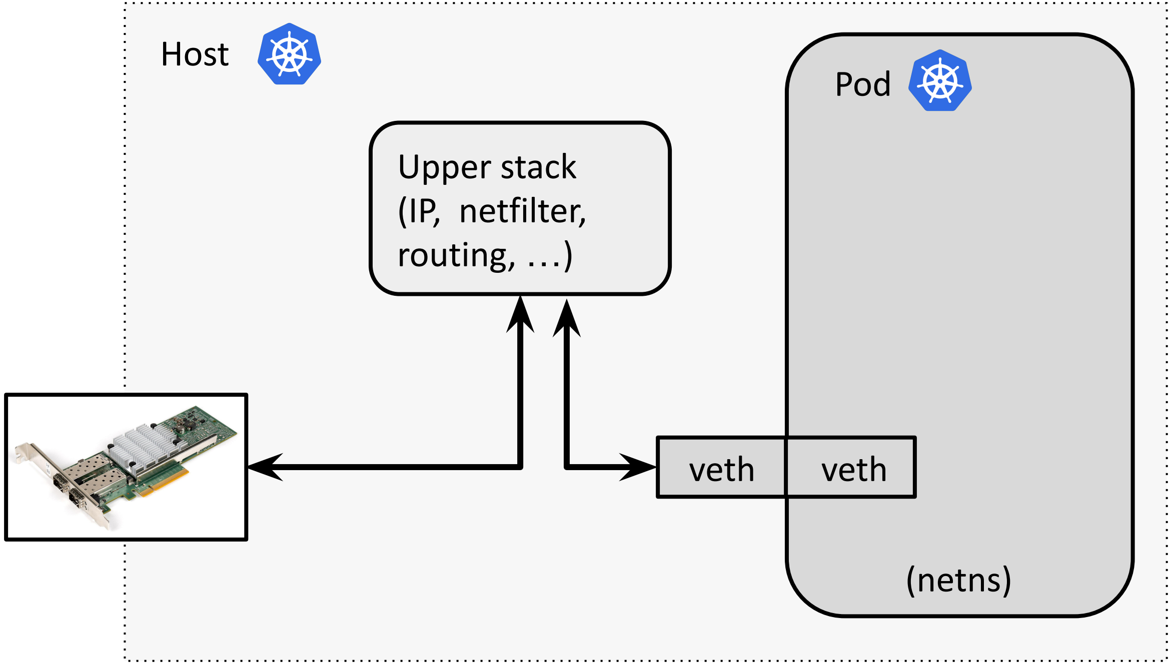
Nikolay Aleksandrov, Isovalent

Goal of this talk

How can we leverage BPF infrastructure & networking features to achieve maximum performance for K8s Pods?

Kubernetes, Pods, CNI in a nutshell

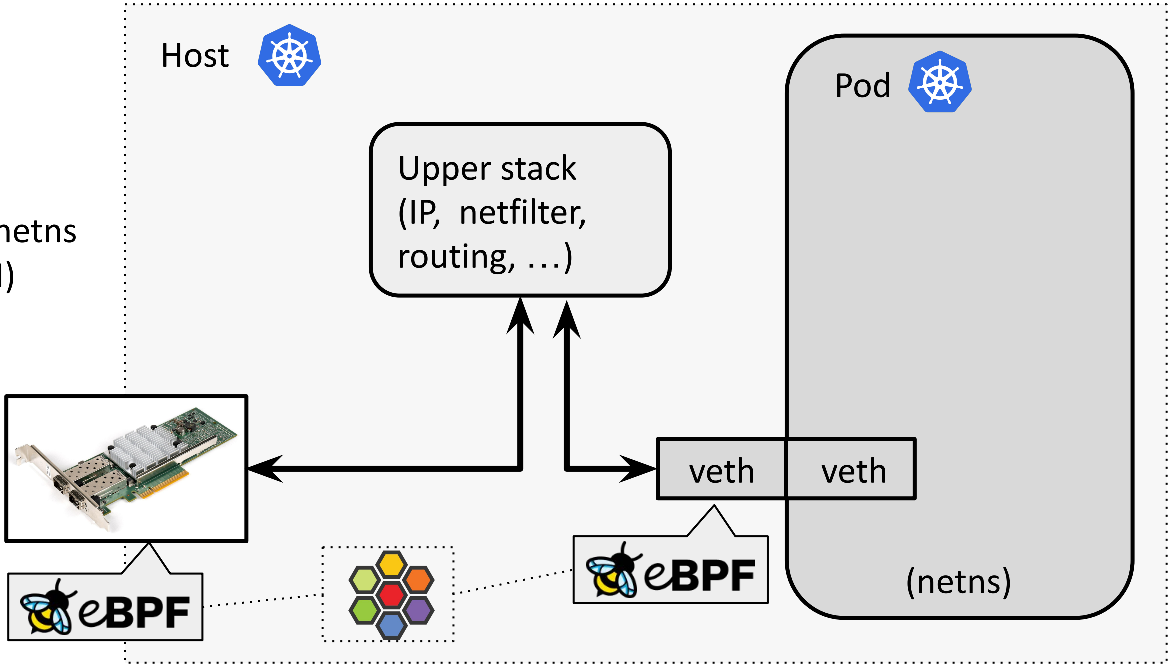
Default Case:



Kubernetes, Pods, CNI in a nutshell

Cilium as CNI:

- Setup netdevs and move to netns
- IP & route assignment (IPAM)
- BPF datapath
- Features on top via BPF:
  - Policy enforcement
  - Load-balancing
  - Bandwidth management
  - etc



Gregg: Computing Performance: What's On the Horizon

My Prediction: OS performance

Linux: increasing complexity & worse perf defaults

- Becomes so complex that it takes an OS team to make it perform well. This assumes that the defaults rot, because no perf teams are running the defaults anymore to notice (e.g., high-speed network engineers configure XDP and QUIC, and aren't looking at defaults with TCP). A bit more room for a lightweight kernel (e.g., BSD) with better perf defaults to compete. Similarities: Oracle DB vs MySQL; MULTICS vs UNIX.

"... becomes so complex that it takes an OS team to make it perform well ..."

SREcon2022: "Computing Performance: What's On the Horizon"

Defaults, and where to go from here ...

Given two K8s nodes with 100Gbit NICs, single flow:

- What’s the default Pod-Pod baseline?
- Where are bottlenecks, how can they be overcome?
- Can we provide better defaults?

Defaults, and where to go from here ...

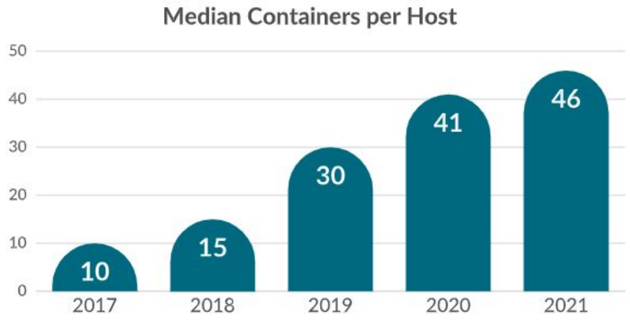
Why bothering with single stream performance?

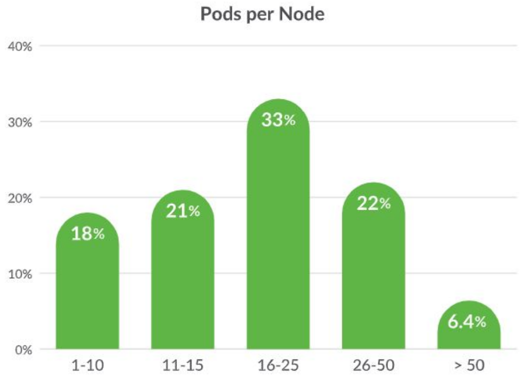
- Coping with growing NIC speeds 100/200/400Gbit
- Big Data/AI/ML and other data intensive workloads
- Generally freeing up resources to save costs

Defaults, and where to go from here ...

Assumptions for our tests:

- K8s worker nodes are **generic** for any kind of workload
  - Large number of users don't custom tune and mostly stick to OS defaults.



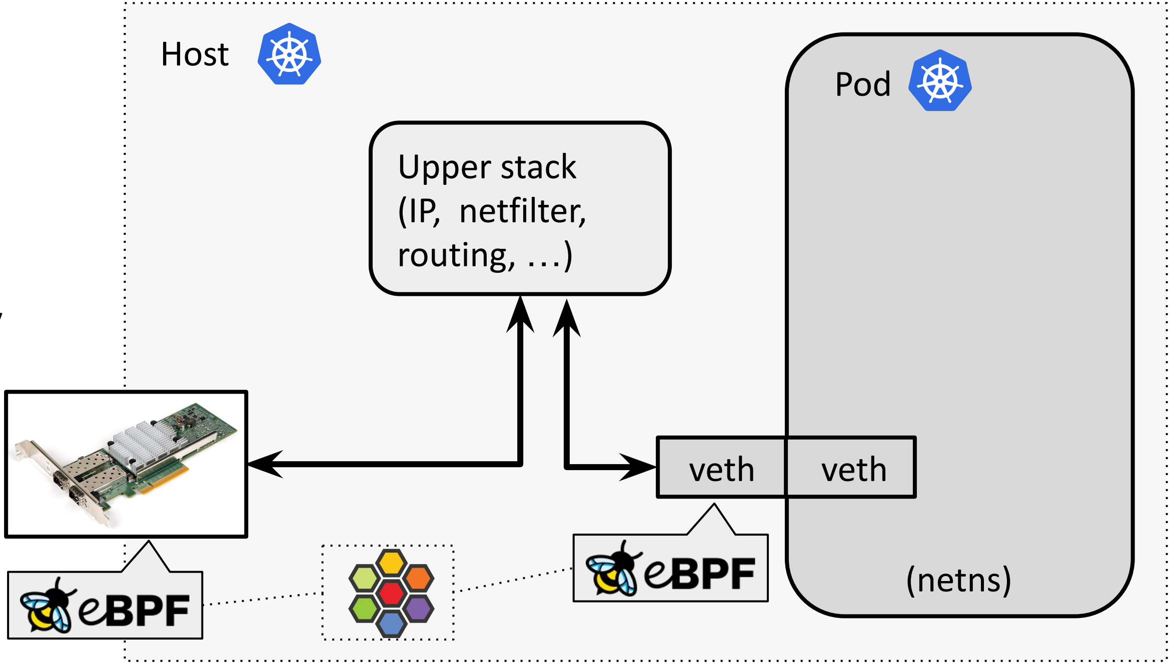


Source: Sysdig 2022 Cloud Native Security and Usage Report

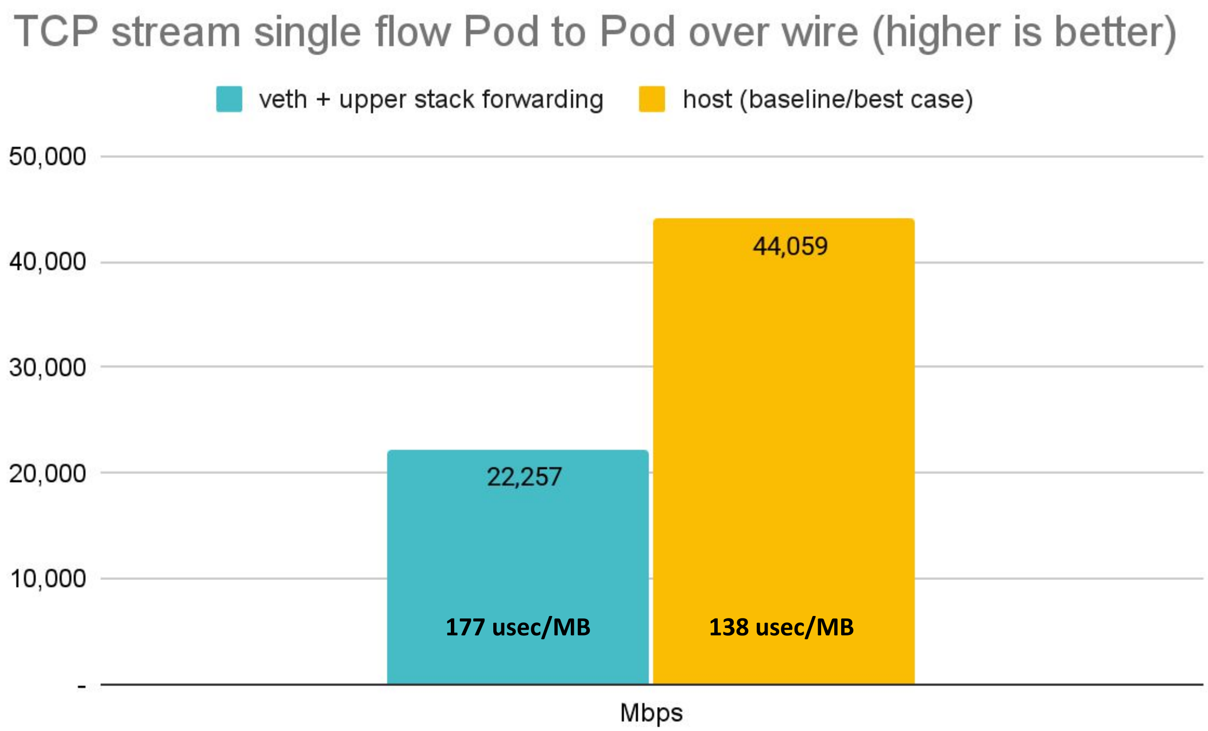
Cilium: Basic/compat setting

Default Case:

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



Default case, results:

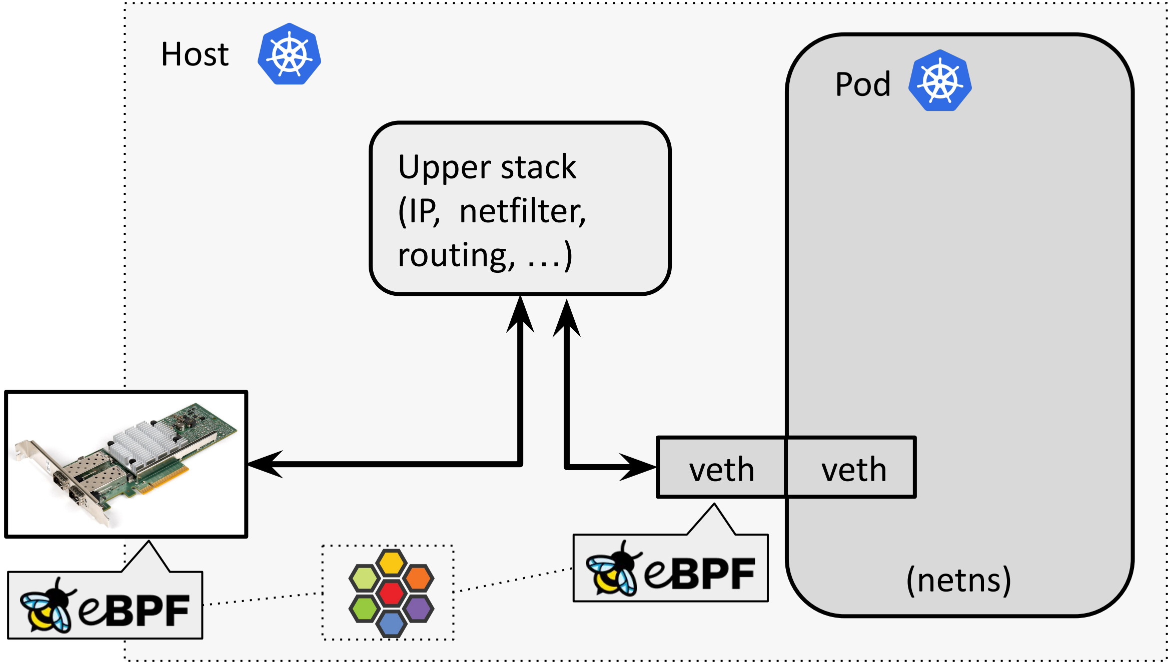


Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO off, 1.5k MTU
Receiver: taskset -a -c <core> tcp_mmap -s (non-zerocopy mode), Sender: taskset -a -c <core> tcp_mmap -H <dst host>

Cilium: BPF host routing

BPF host routing:

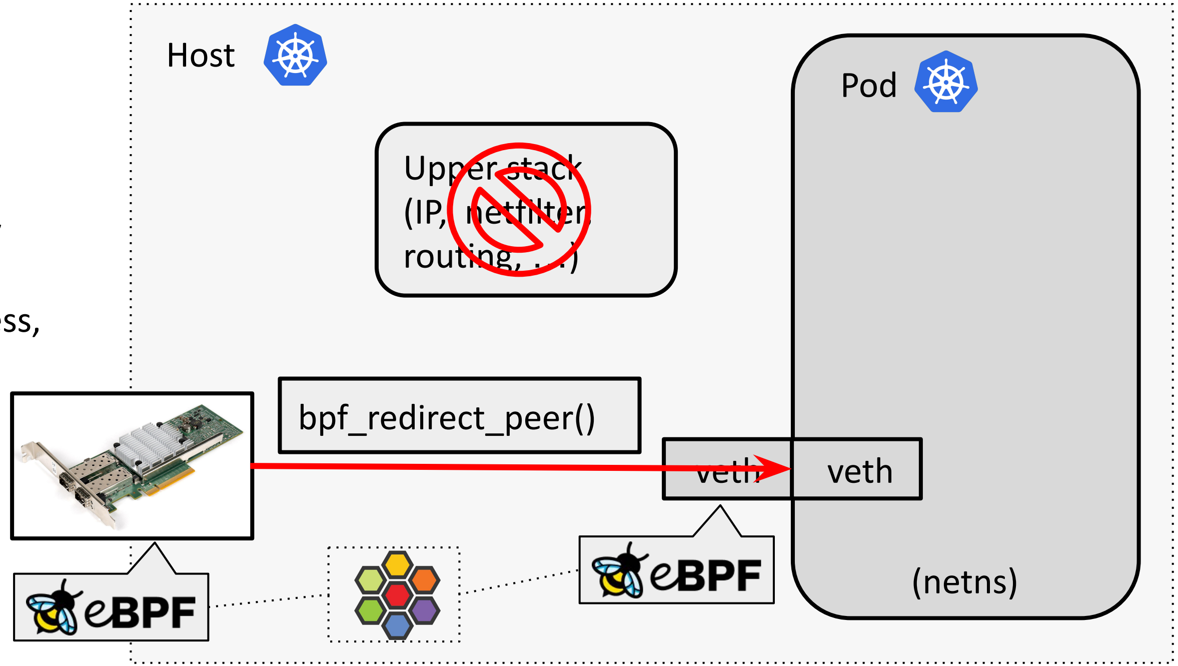
- ~~Routing via upper stack~~



Cilium: BPF host routing

BPF host routing:

- Routing via tc BPF layer only
- Fast netns switch on ingress
- Neighbor resolution on egress,
- If needed bpf_fib_lookup()



Virtual Ethernet Device Optimization with eBPF: https://cilium.io/blog/2020/11/10/cilium-19/#veth

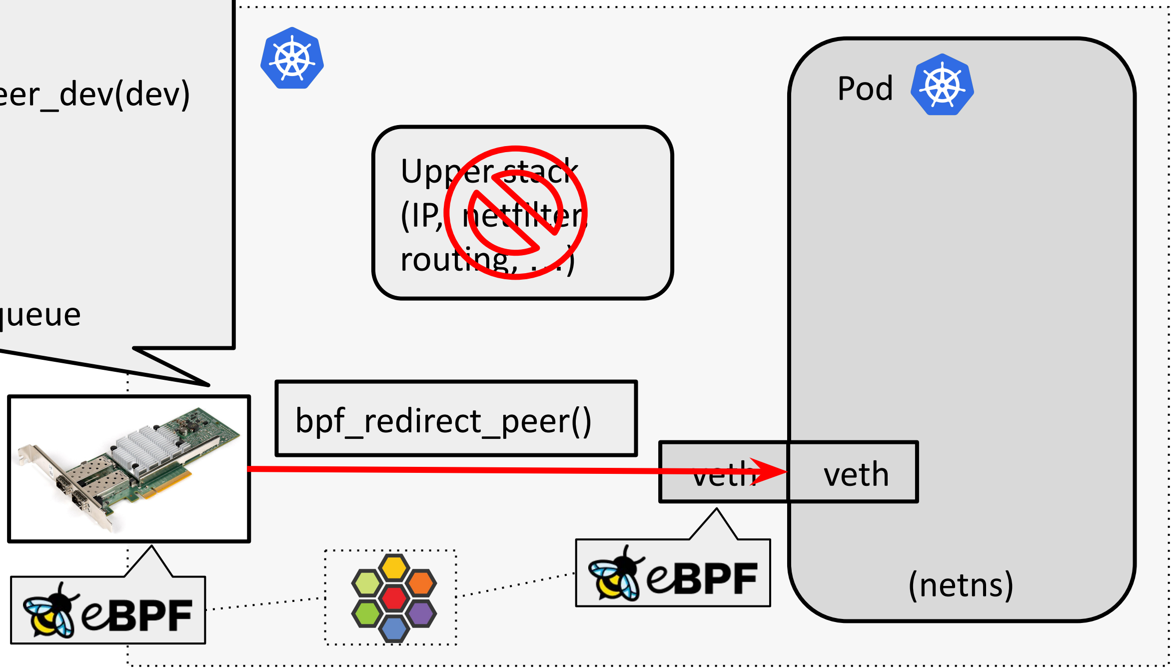
Cilium: BPF host routing

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

BPF h

- Rou
- Fast
- Neig
- If needed bpf_fib_lookup()

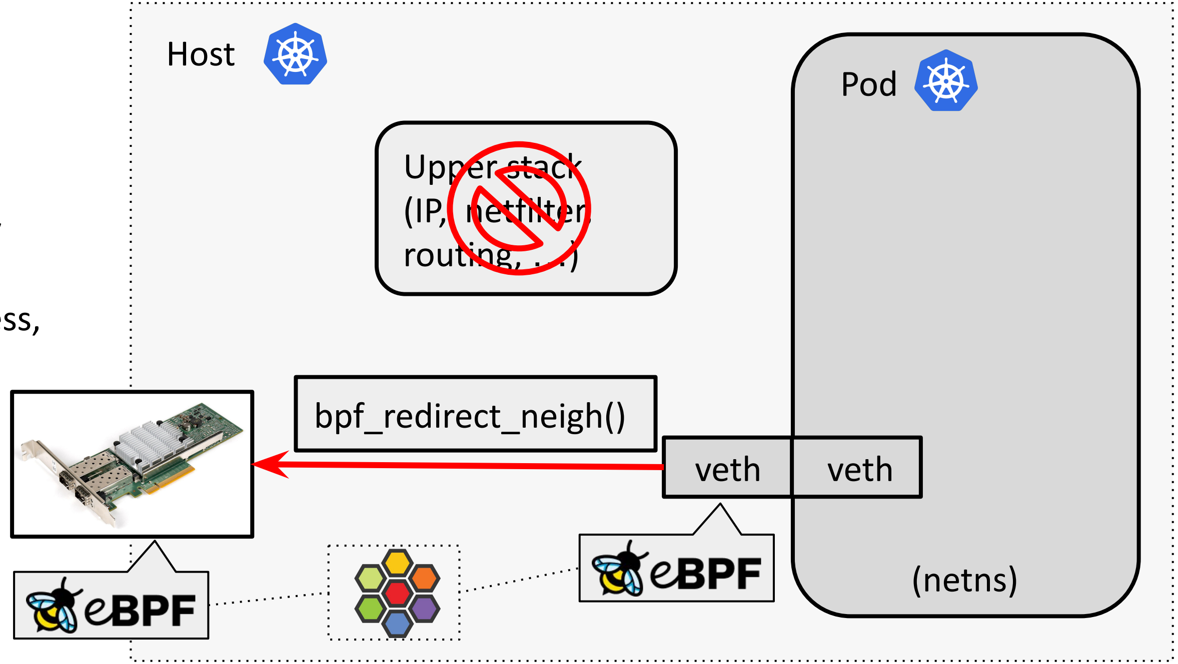


Virtual Ethernet Device Optimization with eBPF: https://cilium.io/blog/2020/11/10/cilium-19/#veth

Cilium: BPF host routing

BPF host routing:

- Routing via tc BPF layer only
- Fast netns switch on ingress
- Neighbor resolution on egress,
- If needed bpf_fib_lookup()



Virtual Ethernet Device Optimization with eBPF: https://cilium.io/blog/2020/11/10/cilium-19/#veth

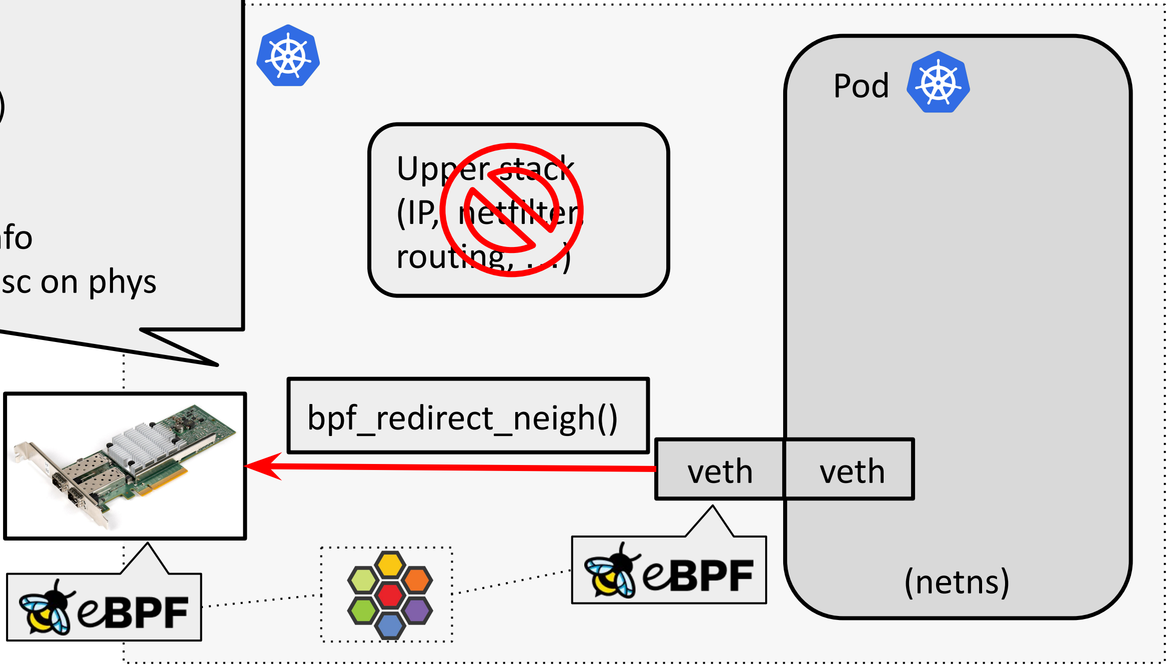
Cilium: BPF host routing

BPF h

- Rou
- Fast
- Neig
- If needed bpf_fib_lookup()

Internals:

ip_route_output_flow()
skb_dst_set()
ip_finish_output2()
- fills in neighbor (L2) info
- retains skb->sk till Qdisc on phys

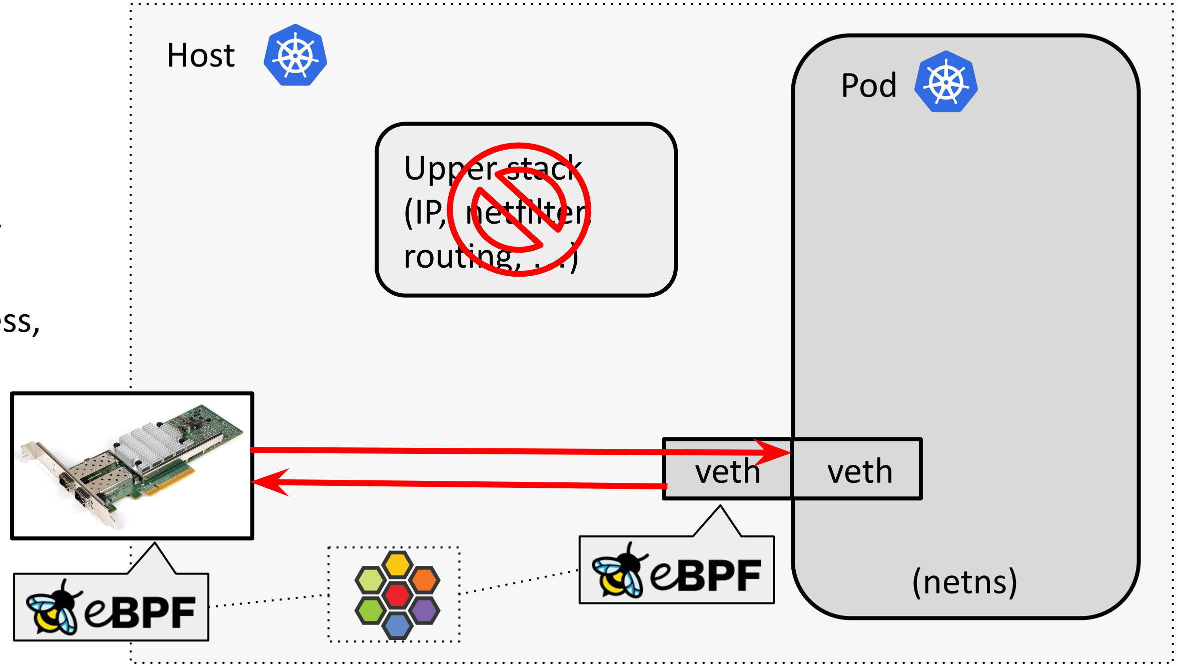


Virtual Ethernet Device Optimization with eBPF: https://cilium.io/blog/2020/11/10/cilium-19/#veth

Cilium: BPF host routing

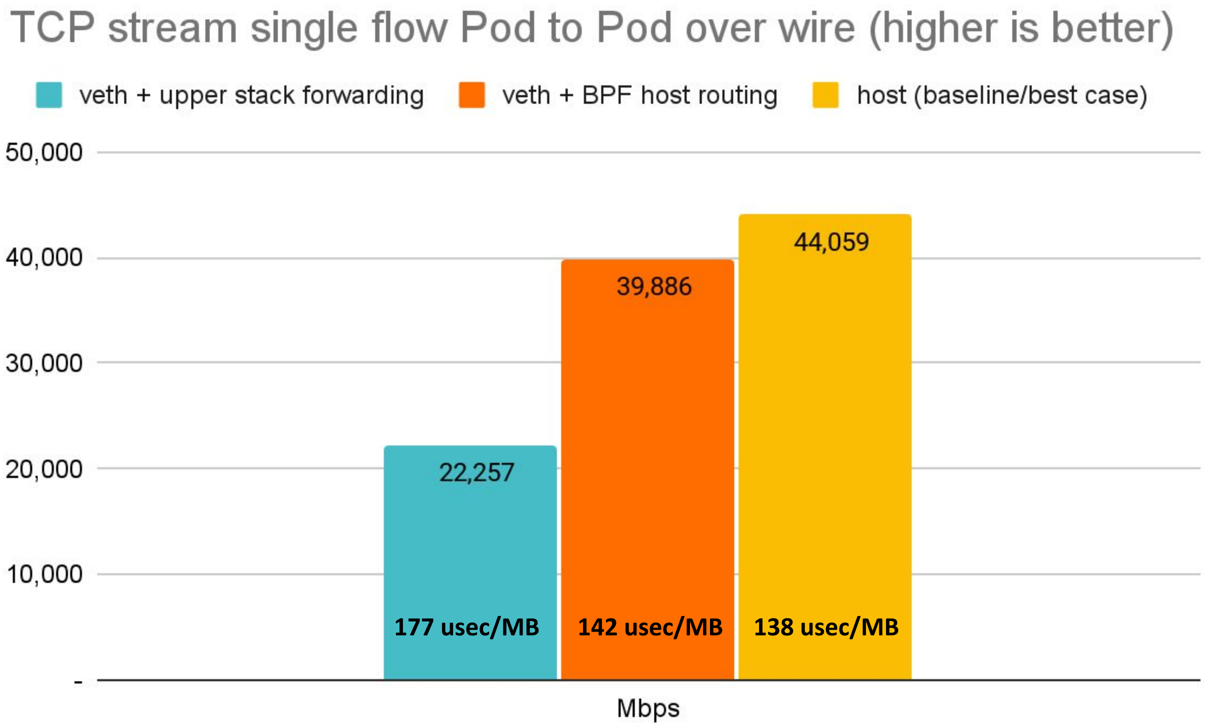
BPF host routing:

- Routing via tc BPF layer only
- Fast netns switch on ingress
- Neighbor resolution on egress,
- If needed bpf_fib_lookup()



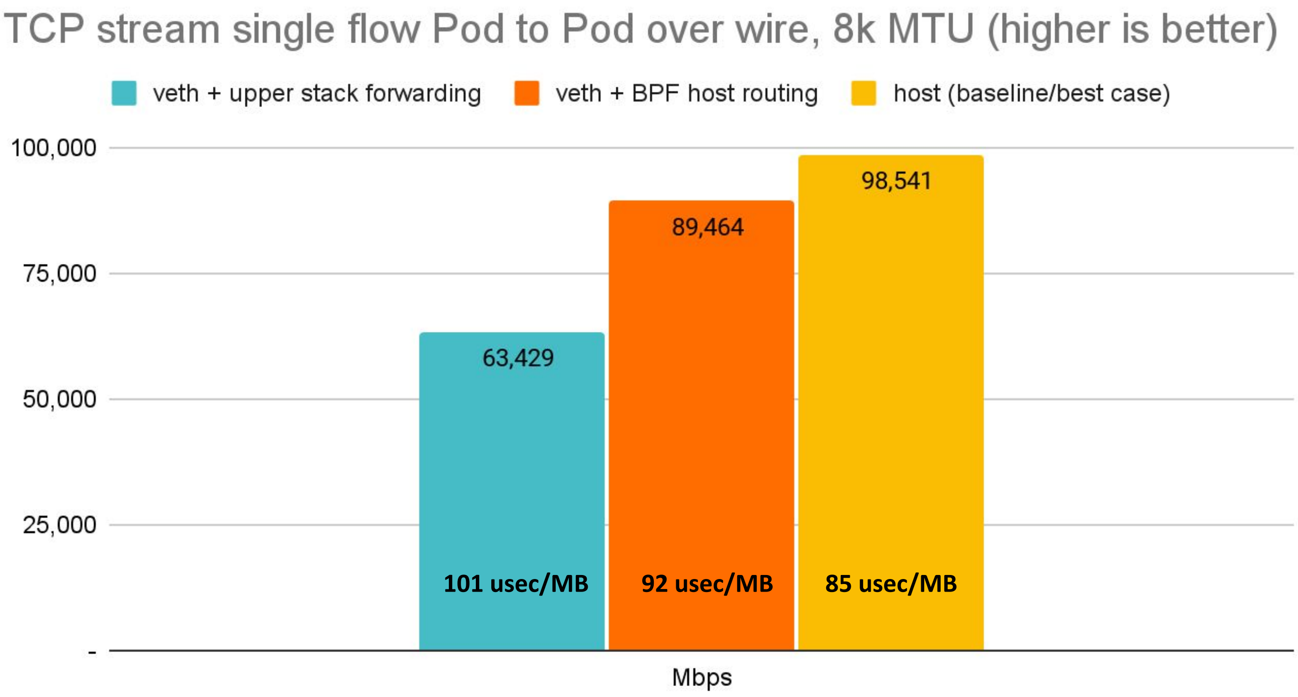
Virtual Ethernet Device Optimization with eBPF: https://cilium.io/blog/2020/11/10/cilium-19/#veth

BPF host routing case, results:



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, 1.5k MTU
Receiver: taskset -a -c <core> tcp_mmap -s (non-zerocopy mode), Sender: taskset -a -c <core> tcp_mmap -H <dst host>

BPF host routing case, with 8k* MTU:



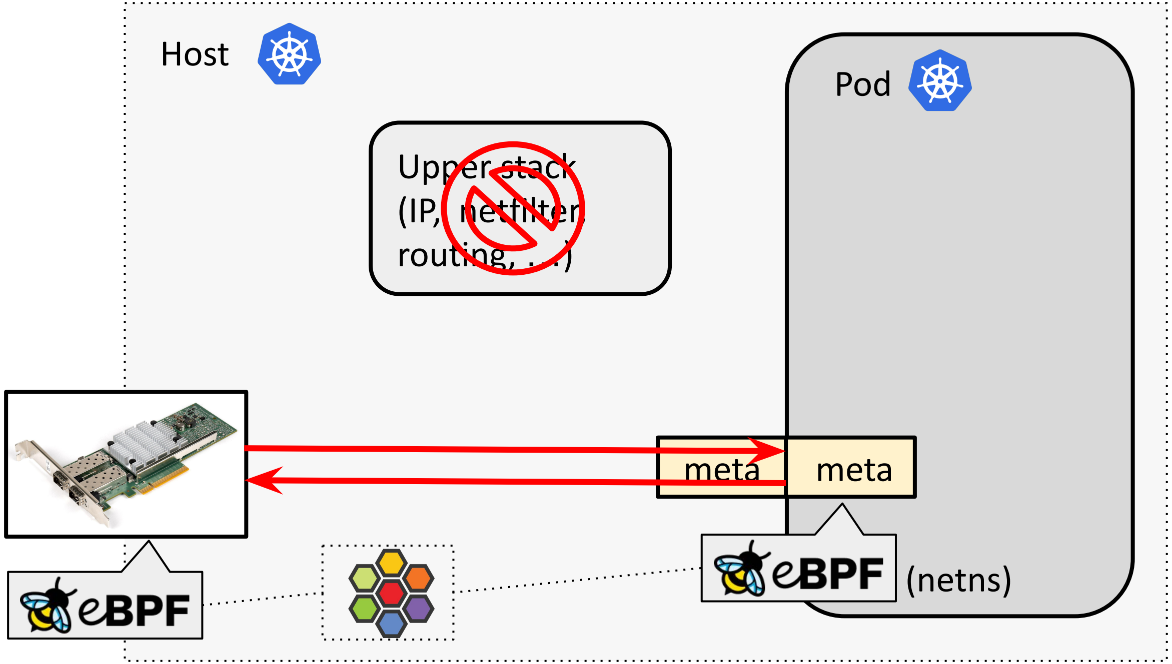
* 8264 MTU for data page alignment in GRO

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: meta devices for BPF

BPF host routing +
meta devices:

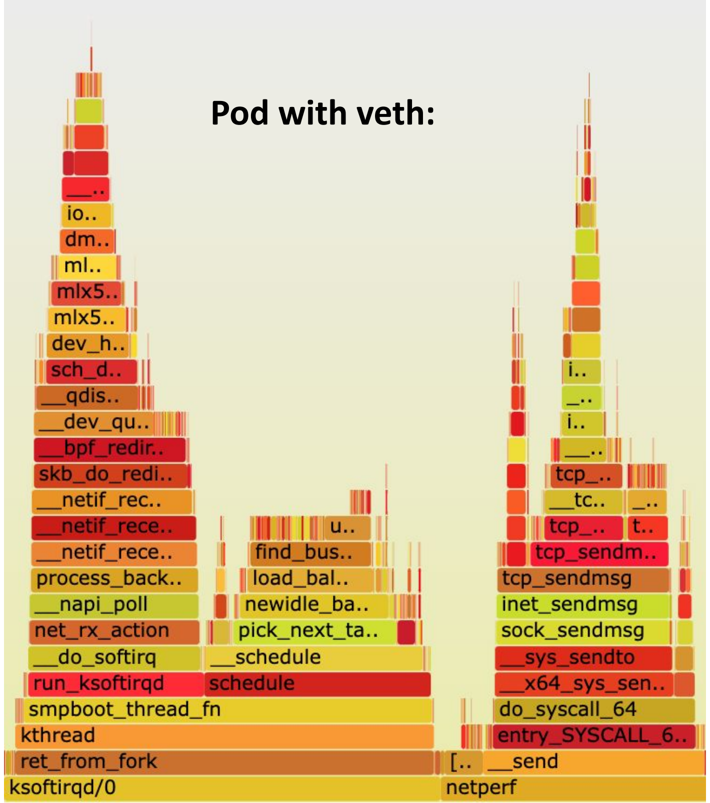
- Routing via tc BPF layer only
- Fast netns switch on ingress
- Fast netns switch on egress
  - BPF prog part of device
    - tc BPF moves into device
  - Changeable only from
    host-side (Cilium)



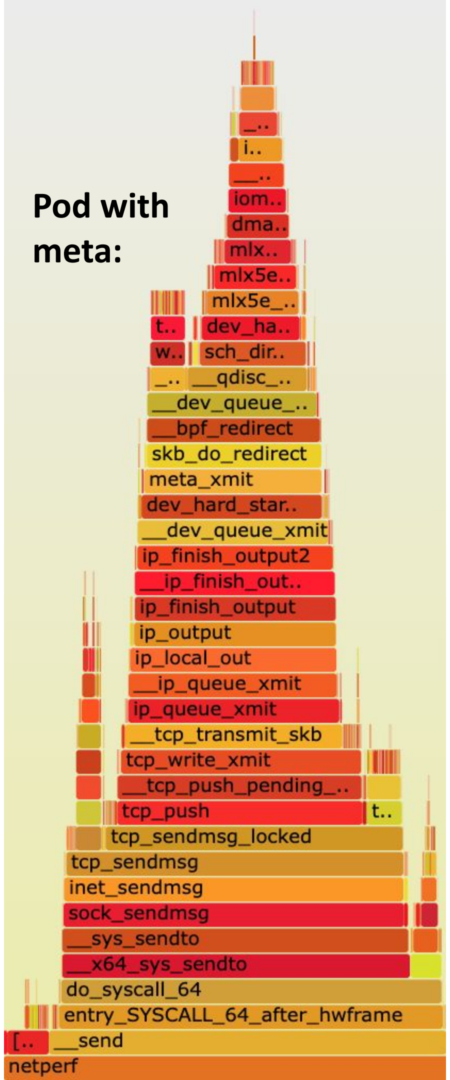
meta prototype: https://github.com/cilium/linux/commits/pr/dev-meta

Cilium: meta devices for BPF

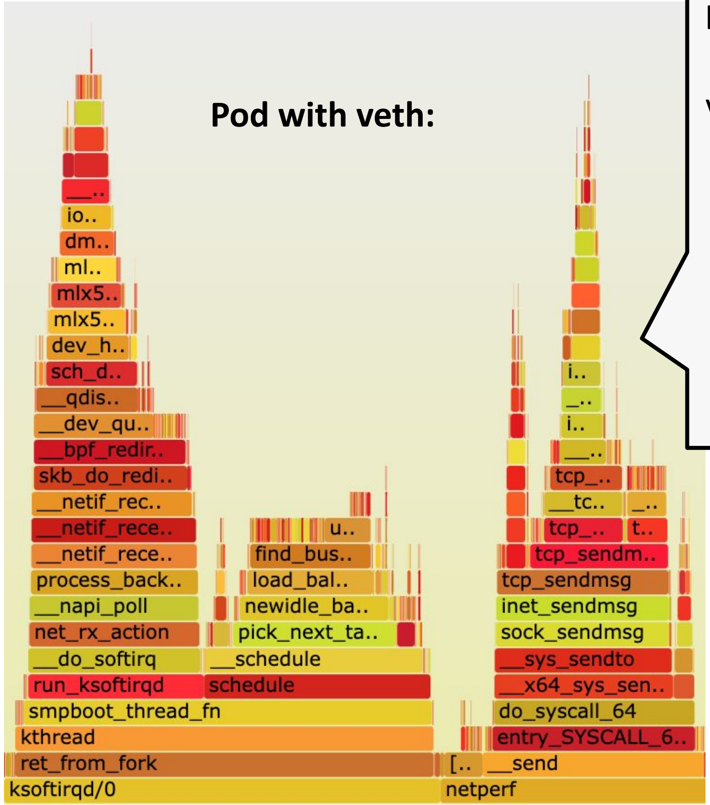
Pod with veth:

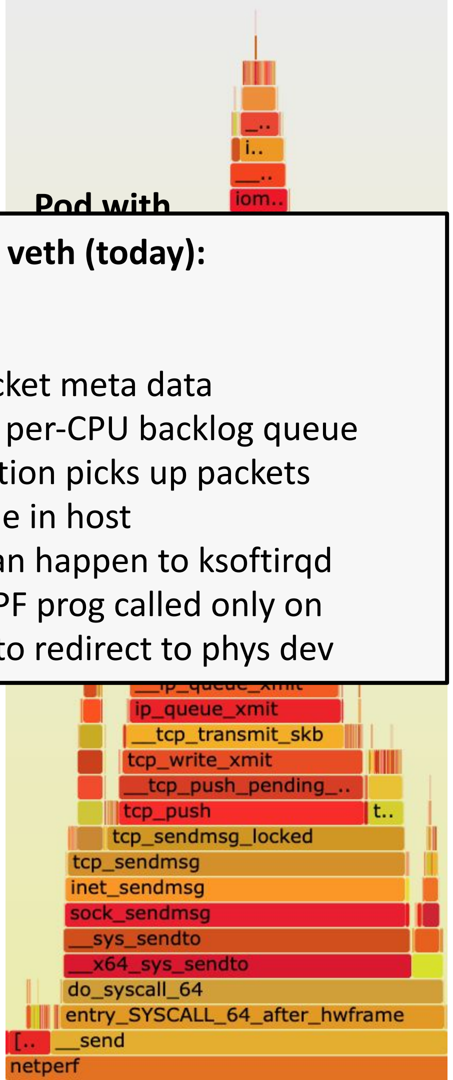


Pod with meta:



Cilium: meta devices for BPF

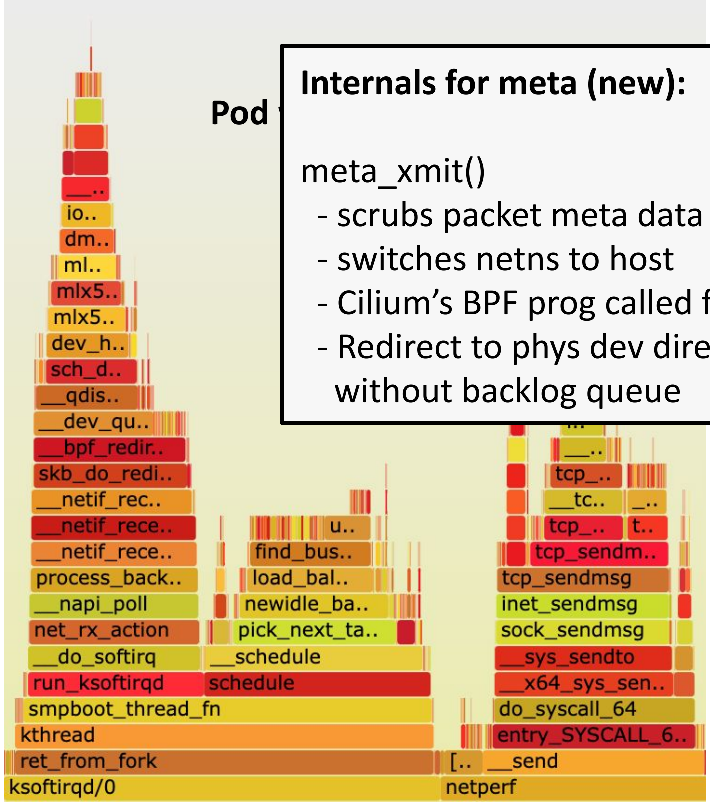




Internals for veth (today):

veth_xmit()
- scrubs packet meta data
- enques to per-CPU backlog queue
- net_rx_action picks up packets from queue in host
- deferral can happen to ksoftirqd
- Cilium's BPF prog called only on tc ingress to redirect to phys dev

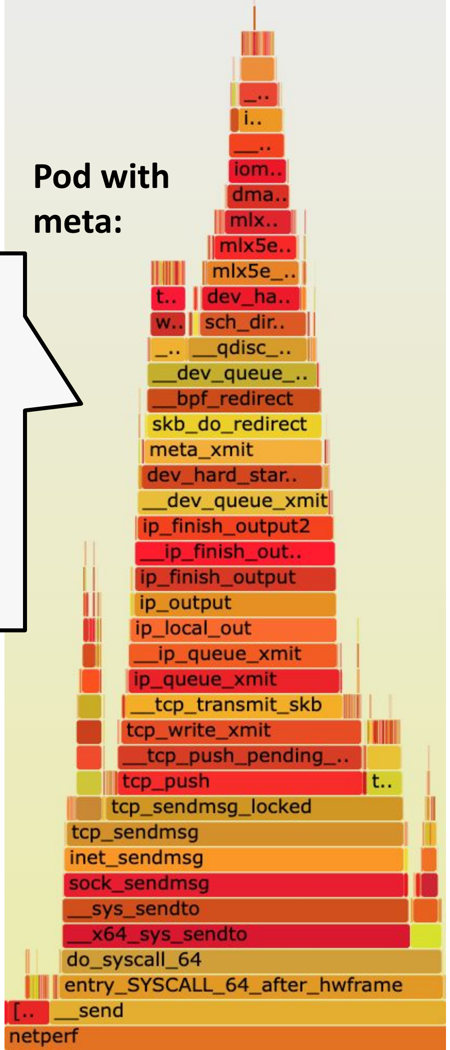
Cilium: meta devices for BPF



Internals for meta (new):

meta_xmit()
- scrubs packet meta data
- switches netns to host
- Cilium’s BPF prog called for meta
- Redirect to phys dev directly without backlog queue

Pod with meta:



meta netdevs

Less is more, ~500 LoCs for the device driver

“meta” given flexibility to implement driver business logic fully in BPF.

Compatibility with tc BPF programs so that for newer kernels they can be migrated easily into meta device.

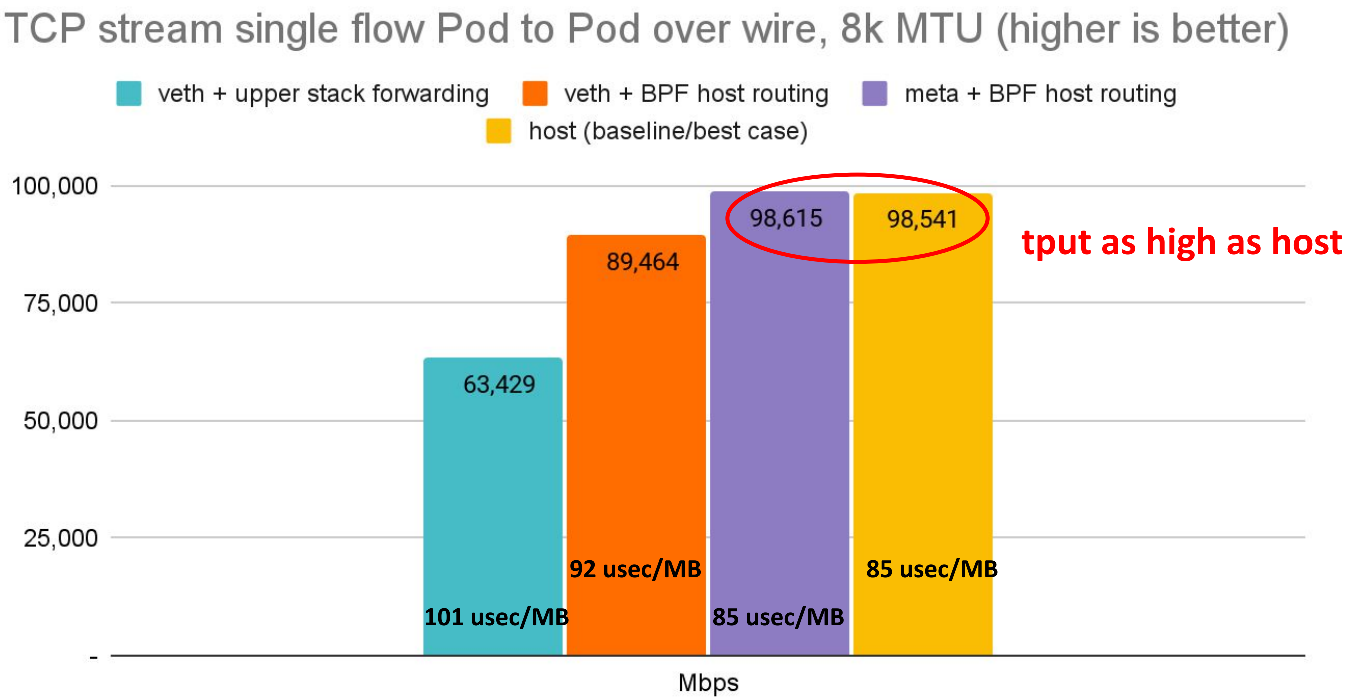
Does not import all the complexity around multi-queue / XDP handling as in veth.

Could operate as single or paired device mode.

```
static netdev_tx_t meta_xmit(struct sk_buff *skb, struct net_device *dev)
{
	struct meta *meta = netdev_priv(dev);
	struct net_device *peer;
	struct bpf_prog *prog;

	rcu_read_lock();
	peer = rcu_dereference(meta->peer);
	if (unlikely(!peer || skb_orphan_frags(skb, GFP_ATOMIC)))
		goto drop;

	meta_scrub_minimum(skb);
	skb->dev = peer;

	prog = rcu_dereference(meta->prog);
	if (unlikely(!prog))
		goto drop;
	switch (bpf_prog_run(prog, skb)) {
	case META_OKAY:
		skb->protocol = eth_type_trans(skb, skb->dev);
		skb_postpull_rcsum(skb, eth_hdr(skb), ETH_HLEN);
		__netif_rx(skb);
		break;
	case META_REDIRECT:
		skb_do_redirect(skb);
		break;
	case META_DROP:
	default:
drop:
		kfree_skb(skb);
		break;
	}
	rcu_read_unlock();
	return NETDEV_TX_OK;
}
```

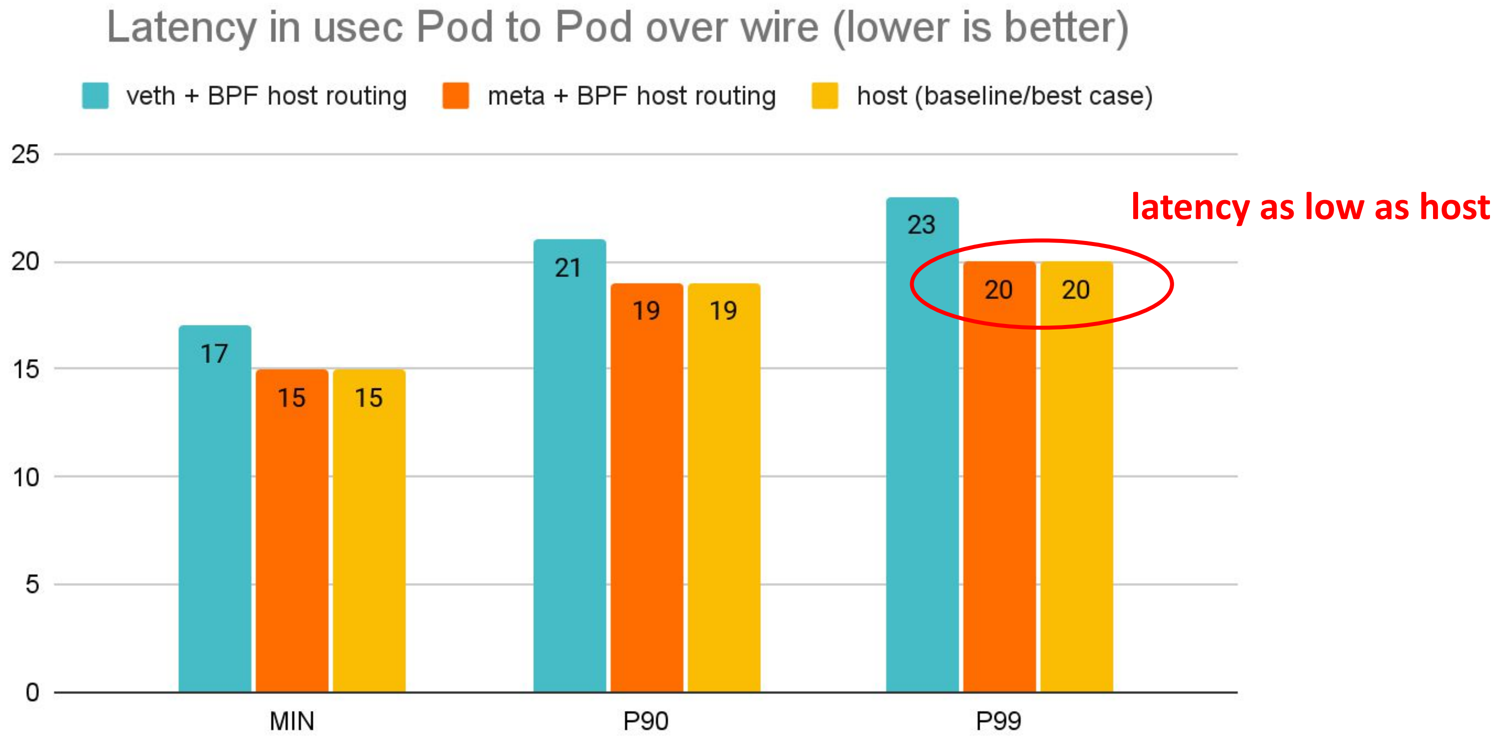
meta prototype: https://github.com/cilium/linux/commits/pr/dev-meta

meta + BPF host routing case, results:



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>

meta + BPF host routing case, results:

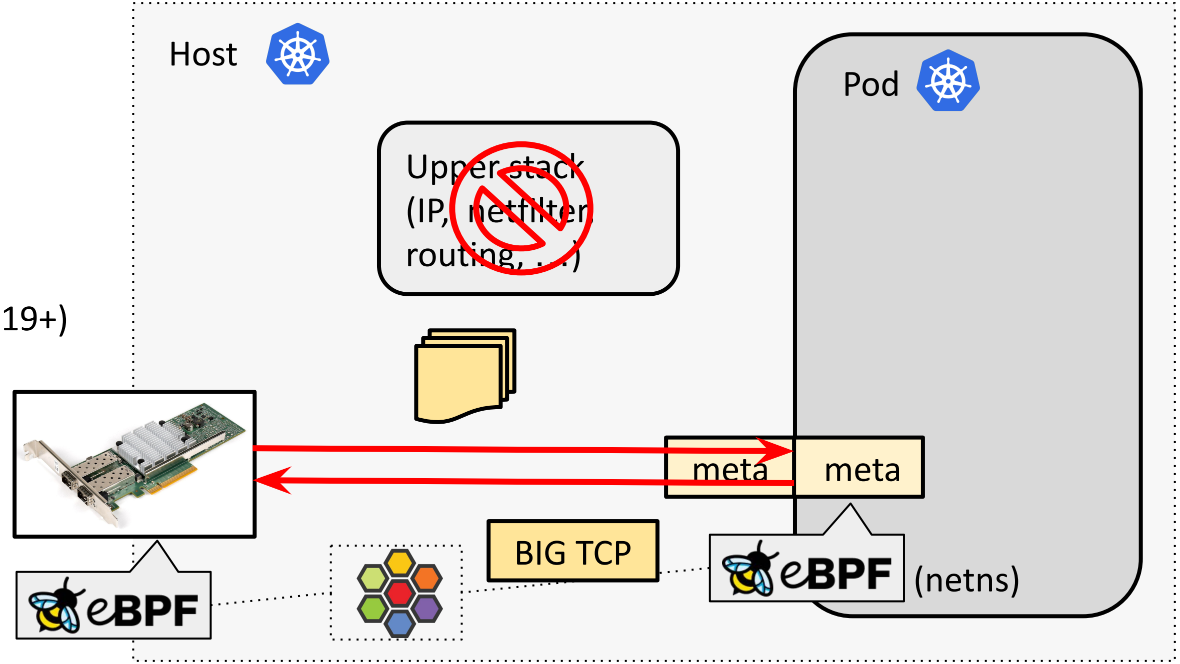


Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, striding mode, LRO off
netperf -t TCP_RR -H <remote pod> -- -O MIN_LATENCY,P90_LATENCY,P99_LATENCY,THROUGHPUT

Cilium: Can we push even further? BIG TCP!

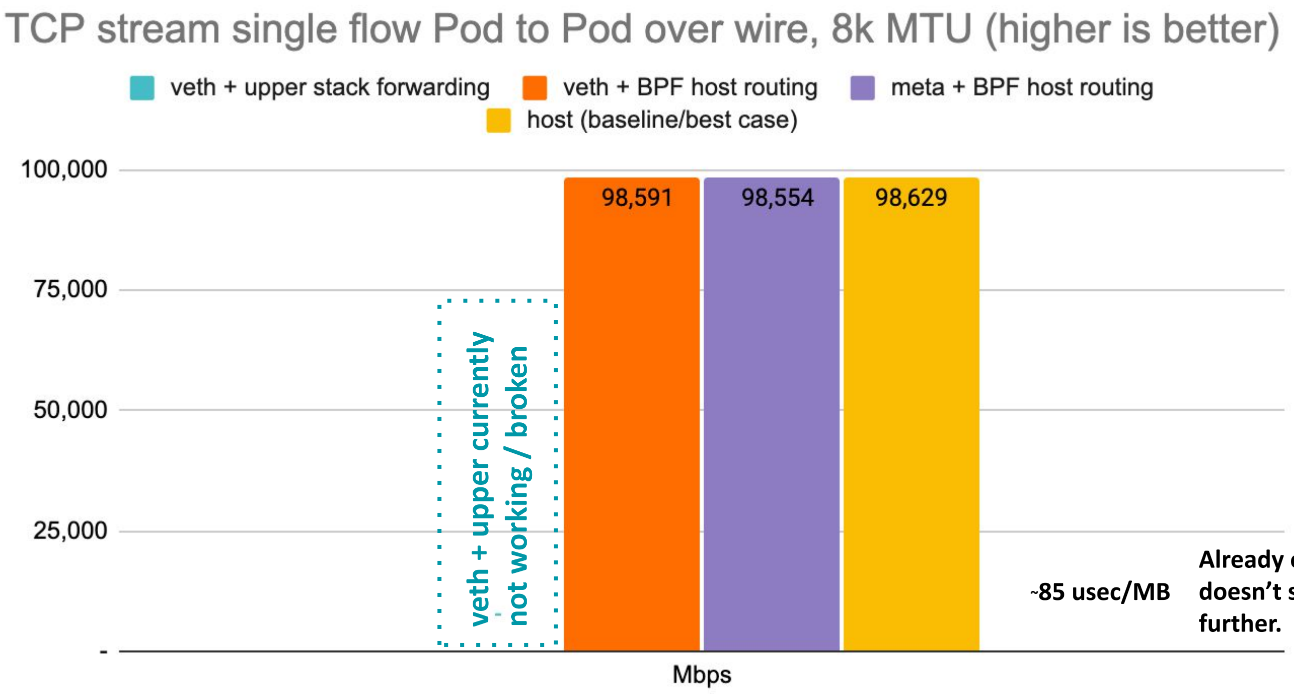
BPF host routing +
meta devices +
BIG TCP:

- Currently only for IPv6* (v5.19+)
- More aggressive GRO/GSO batching with HBH header
- Supported by Cilium 1.13



Cilium + BIG TCP, KubeCon North America 2022: https://kccncna2022.sched.com/event/182DB, * IPv4 BIG TCP merged this week for v6.3+

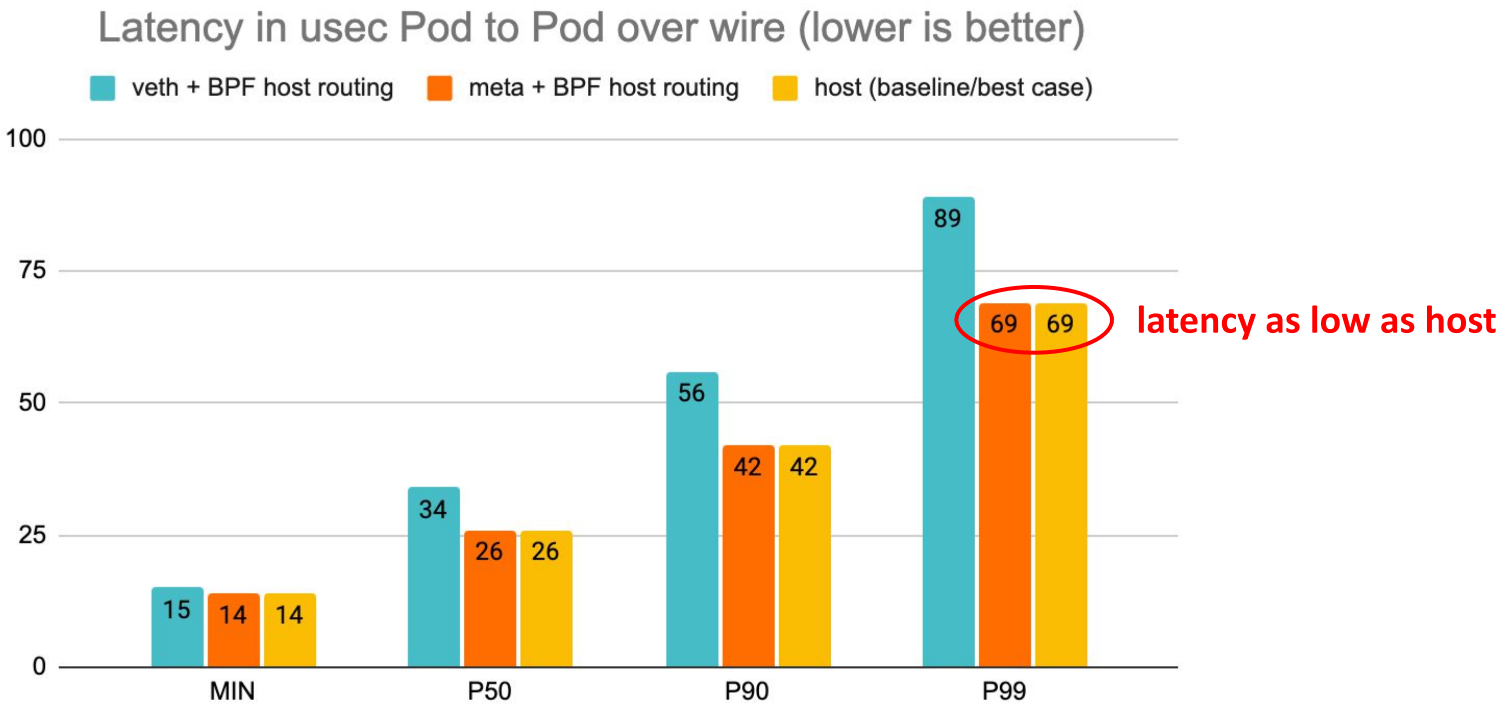
BIG TCP + BPF host routing case, results:



Already on limit, BIG TCP doesn't shrink usec/MB further.

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>

BIG TCP + BPF host routing case, results:



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
netperf -t TCP_RR -H <remote pod> -- -O MIN_LATENCY,P90_LATENCY,P99_LATENCY,THROUGHPUT

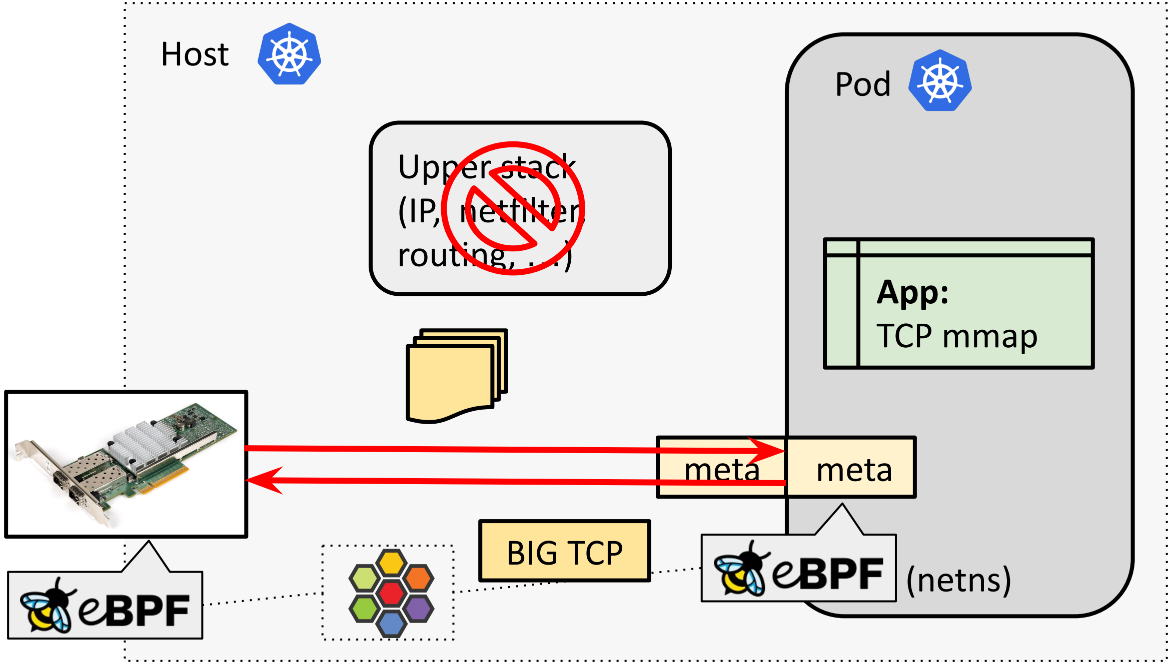
Remaining biggest offender is copy to user

```
- read
   - 63.41% entry_SYSCALL_64_after_hwframe
      - 63.36% do_syscall_64
         - 63.20% __x64_sys_read
            - 63.18% ksys_read
               - 63.01% vfs_read
                  - 62.92% new_sync_read
                     - 62.85% sock_read_iter
                        - 62.79% sock_recvmsg
                           - 62.77% inet6_recvmsg
                              - 62.65% tcp_recvmsg
                                 - 61.36% tcp_recvmsg_locked
                                    + 58.27% skb_copy_datagram_iter
                                    + 2.62% tcp_cleanup_rbuf
                                   0.56% release_sock
```

Cilium: Can we push even further? BIG TCP + ZC

BPF host routing +
meta devices +
BIG TCP +
TCP mmap?

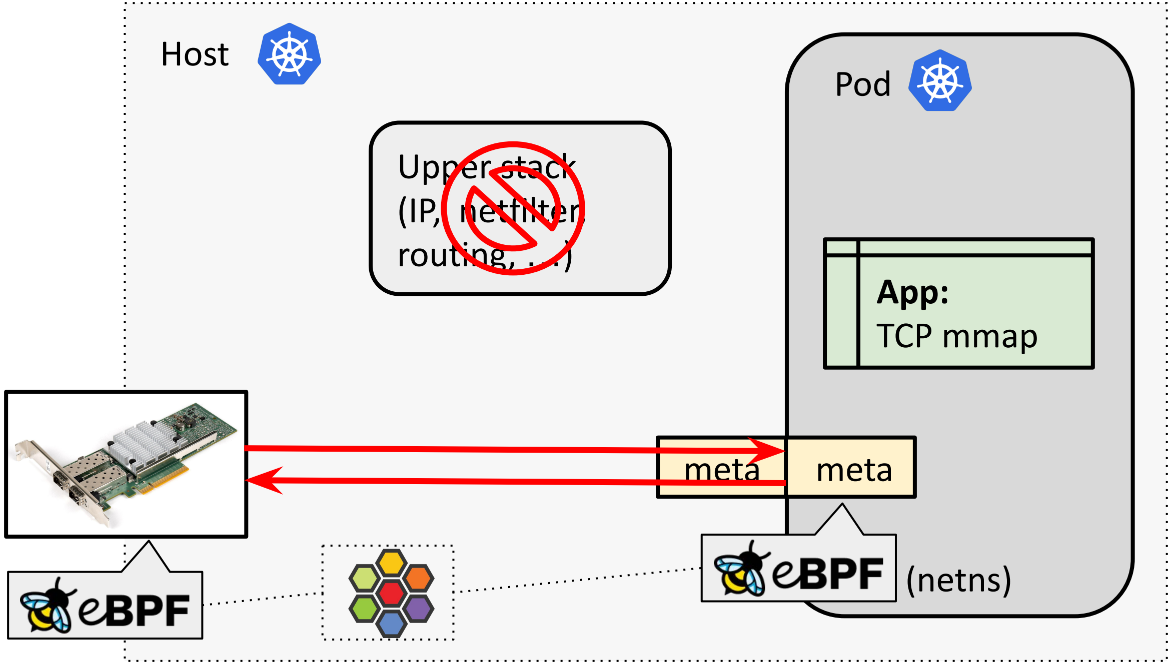
- Currently not possible
- BIG TCP generates frag_list
- TCP ZC works on skb frags[]
- Combining has the highest potential for pushing boundaries further …
  - Let's look at *just* TCP ZC



Cilium: Can we push even further? TCP ZC

BPF host routing +
meta devices +
TCP mmap

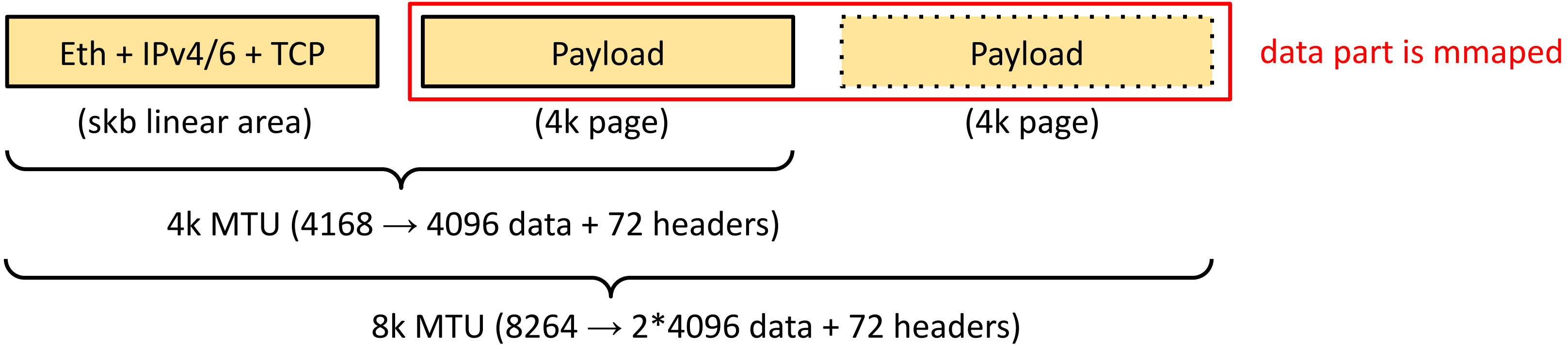
- Not as straightforward
  - Needs app changes for ZC on RX and/or TX
  - Needs driver changes to implement pseudo header/data split if not natively done by HW



mlx5 header split: https://github.com/cilium/linux/tree/test/zc-hdsplit

TCP ZC, header split and other caveats

Header/data split:



See Eric's talk for details, e.g. TCP WSCALE needs to be raised to 12 to get aligned RWIN to avoid partially filled pages.

Mileage varies on driver/HW support on header/data split, e.g. we implemented a PoC for mlx5 given not upstream.

Good example application for RX & TX TCP zero-copy is tcp_mmap in networking selftests.

TCP ZC, header split and other caveats

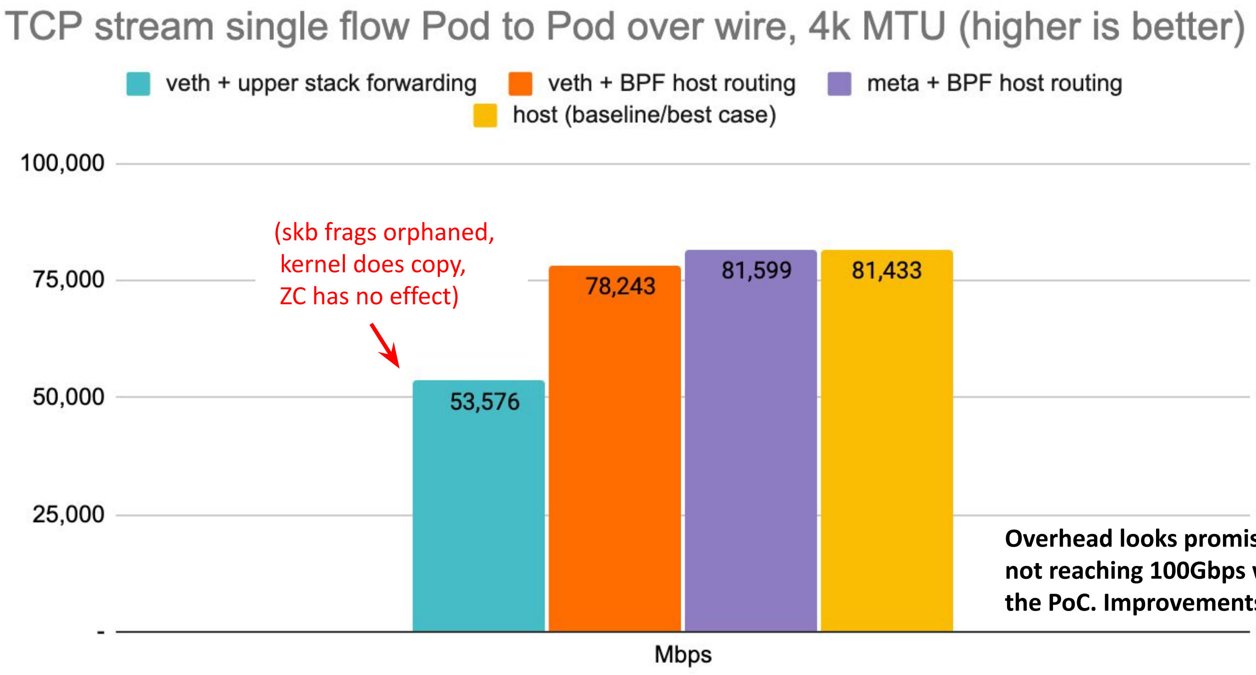
Various settings need to be considered:

- mlx5 (mainly just our PoC): `ethtool --set-priv-flags eth0 rx_striding_rq off`
- MTU is set to 4168 (4k) or 8264 (8k), implicitly affects TCP ADVMSS
- For pinning TCP WSCALE the TCP rmem/wmem must be adapted e.g. `"4096 67108864 134217728"`
- For TX zero-copy optmem needs tuning: `sysctl net.core.optmem_max=1048576`
- Contention/overhead in IOMMU and page clearing: `iommu=off, init_on_alloc=0 init_on_free=0`
- Page recycling from page pool cannot be reused anymore

Header/data split could be a useful addition for ethtool (Windows actually has a config framework for splitting).

TCP zero-copy benefits might be limited if application needs to pull data into cache.

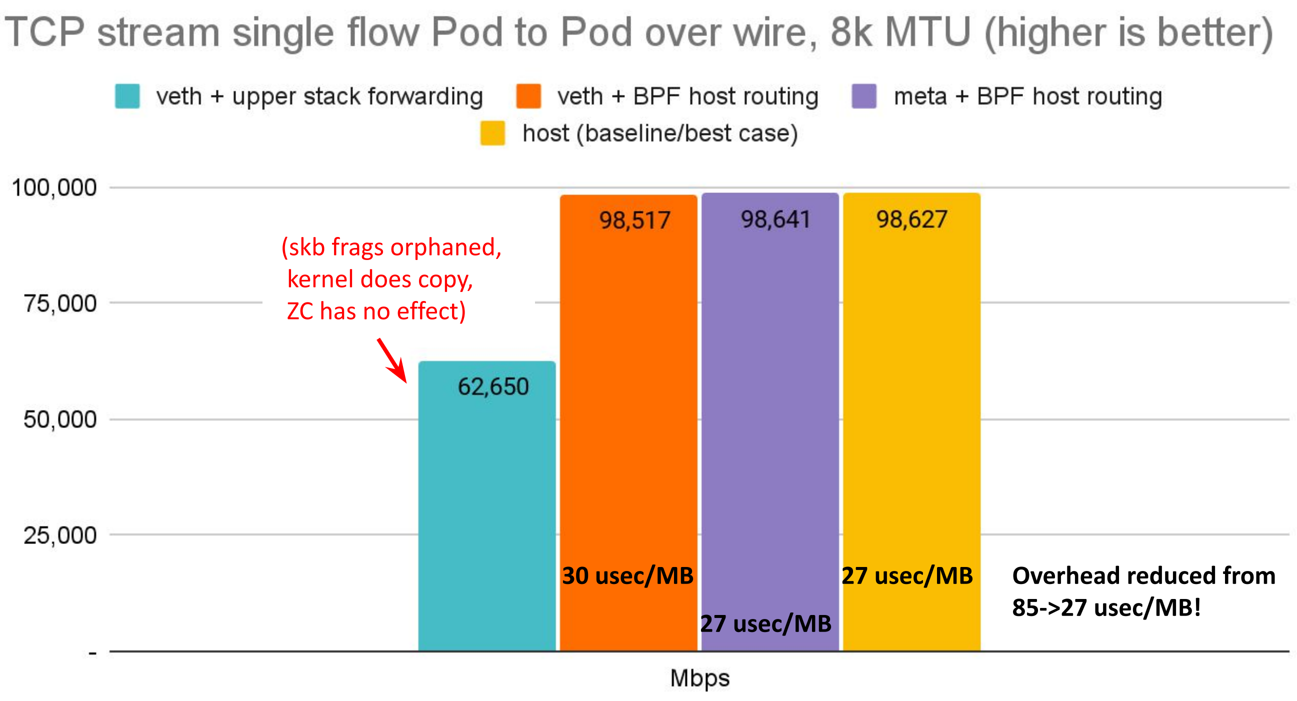
TCP ZC + BPF host routing case, results:



Overhead looks promising, but not reaching 100Gbps with 4k with the PoC. Improvements possible.

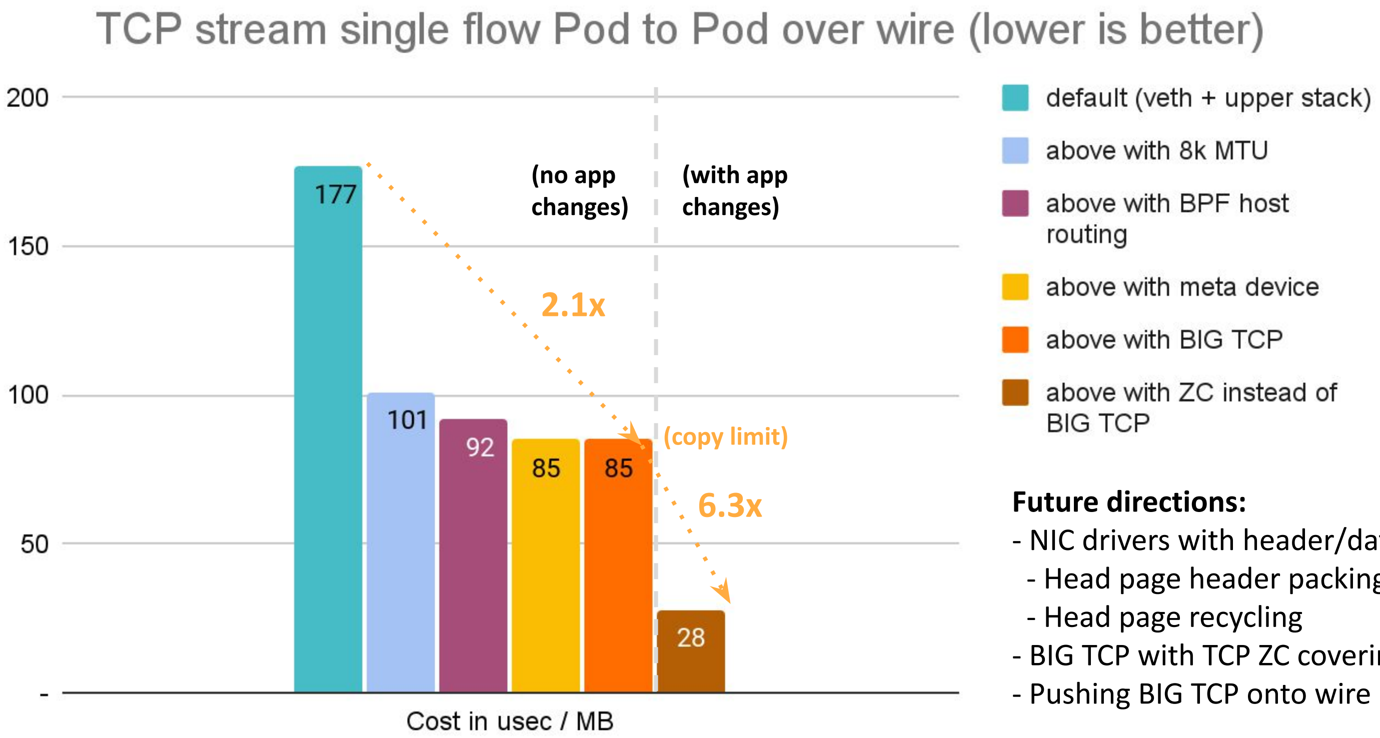
Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver, legacy mode, LRO off, 4168 MTU
Receiver: taskset -a -c <core> tcp_mmap -s -z (zerocopy mode), Sender: taskset -a -c <core> tcp_mmap -H <dst host> -z

TCP ZC + BPF host routing case, results:



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

Recap on defaults and how to reduce cost



Future directions:

- NIC drivers with header/data split setting
  - Head page header packing
  - Head page recycling
- BIG TCP with TCP ZC covering frag_list
- Pushing BIG TCP onto wire if HW supports it

ISOVALENT

Thank you! Questions?

github.com/cilium/cilium

cilium.io

ebpf.io

meta device: github.com/cilium/linux/commits/pr/dev-meta

header/data split: github.com/cilium/linux/commits/test/zc-hdsplit