KubeCon | CloudNativeCon

North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

ISOVALENT

100Gbit/s Clusters with Cilium: Building Tomorrow's Networking Data Plane

Daniel Borkmann, Cilium Team & eBPF co-maintainer

Nikolay Aleksandrov, Cilium Team

Nico Vibert, Cilium Team

What are the challenges with running large-scale data center networks?

Scale

Performance

Operations

WHAT IF...?

IPv6 could address both scale *and* performance requirements

“The Cilium Experiment”

Before looking into the future, let's take a short detour back to 2016 ...

“The Cilium Experiment”

Source: https://www.slideshare.net/ThomasGraf5/cilium-fast-ipv6-container-networking-with-bpf-and-xdp

“The Cilium Experiment”

We started out with IPv6-only networking.

Scalable, flexible, global addressing. No NAT.

“The Cilium Experiment”

Abstracting away from traditional networking models. Building Cilium’s datapath upon eBPF for maximum efficiency.

“The Cilium Experiment”

... until reality kicked in.

“The Cilium Experiment”

IPv6 Status in Kubernetes/Docker

(2016)

- **Kubernetes (CNI): Almost there**
  - Pods are IPv6-only capable as of k8s 1.3.6 (PR23317, PR26438, PR26439, PR26441)
  - Kubeproxy (services) not done yet
- **Docker (libnetwork): Working on it**
  - PR826 - *“Make IPv6 Great Again”*
    Not merged yet

Source: https://www.slideshare.net/ThomasGraf5/cilium-fast-ipv6-container-networking-with-bpf-and-xdp

“The Cilium Experiment”

... and we had to implement IPv4 support, upon popular demand. ;-)

K8s and IPv6

Fast forward 2022, Kubernetes has IPv6 Single (GA v1.18) and IPv4/IPv6 Dual Stack support (GA v1.23).

K8s and IPv6

Hyper-scalers have also made progress integrating IPv6 into their services, although most of it Dual Stack.

K8s and IPv6

Managed K8S offerings (AKS, EKS and GKE) all offer various levels of IPv6 support: AKS (Dual Stack in Preview), GKE (Dual Stack) and EKS (Single Stack).

K8s and IPv6

User needs: More IPAM flexibility and headroom via IPv6 Single Stack, in particular, on larger clusters with lots of IP churn.

K8s and IPv6

Dual Stack regarded as transitional state.

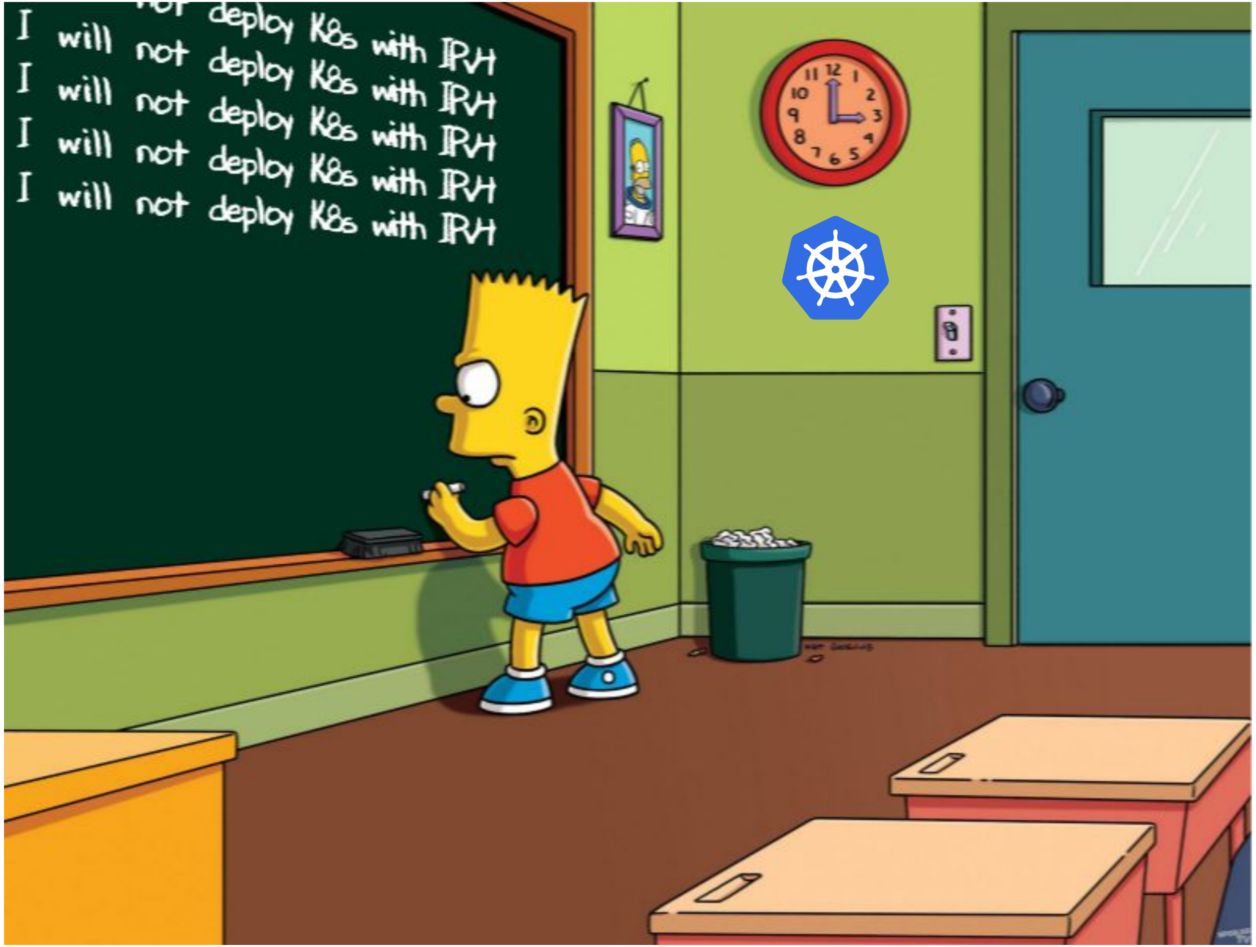
IPv6 end-to-end desired state in general.

Avoids complexity from IPv4 *and* IPv6.

KubeCon EU 2022, Dual Stack adoption advisory panel: https://sched.co/ytt0

K8s and IPv6

Approach: Building out islands of IPv6 Single Stack on-prem clusters as clean-slate for successive application/service migration.


I will not deploy K8s with IPv4
I will not deploy K8s with IPv4
I will not deploy K8s with IPv4
I will not deploy K8s with IPv4

K8s and IPv6

However, interfacing with IPv4 will very likely remain for now. (Unless air-gapped, or lucky with external dependencies.)

K8s and IPv6

2022 “fun” fact (or better, sad state):

Some statistics

Out of the top 1000 Alexa sites, only 466 has IPv6 enabled, and 843 of them use nameservers with IPv6 enabled.
Of the total 902708 sites only 34.6% of them have IPv6. This is a huge shame!

Source: https://whynoipv6.com/

K8s and IPv6

Lots of ecosystem bumps on the road, for example, GitHub is still not IPv6 accessible.

K8s and IPv6

IPv6 support for cloning Git repositories #10539

Unanswered aparcar asked this question in **General**

aparcar on 25 Jan

Hi, I'm surprised I didn't find an existing discussion with this topic. Some services like github-releases.githubusercontent.com or user pages do support IPv6, however the webpage (github.com) itself, including cloning of git repositories, does not work.

Is there a public roadmap on enabling IPv6 for GitHubs very core business, distributing Git repositories? If I'm wrong and there is already IPv6 support, please guide me.

The same issues exists for api.github.com and thereby making CLIs unusable on IPv6 only connections.

↑ 238 · 👍 114 · 🎉 9 · 😕 12 · ❤️ 15 · 🚀 7 · 👀 13

Source: https://github.com/community/community/discussions/10539

K8s and IPv6

So, in a modern, IPv6-only K8s cluster,

how should one deal with “legacy” IPv4?

Enter: NAT46 and NAT64

K8s and IPv6

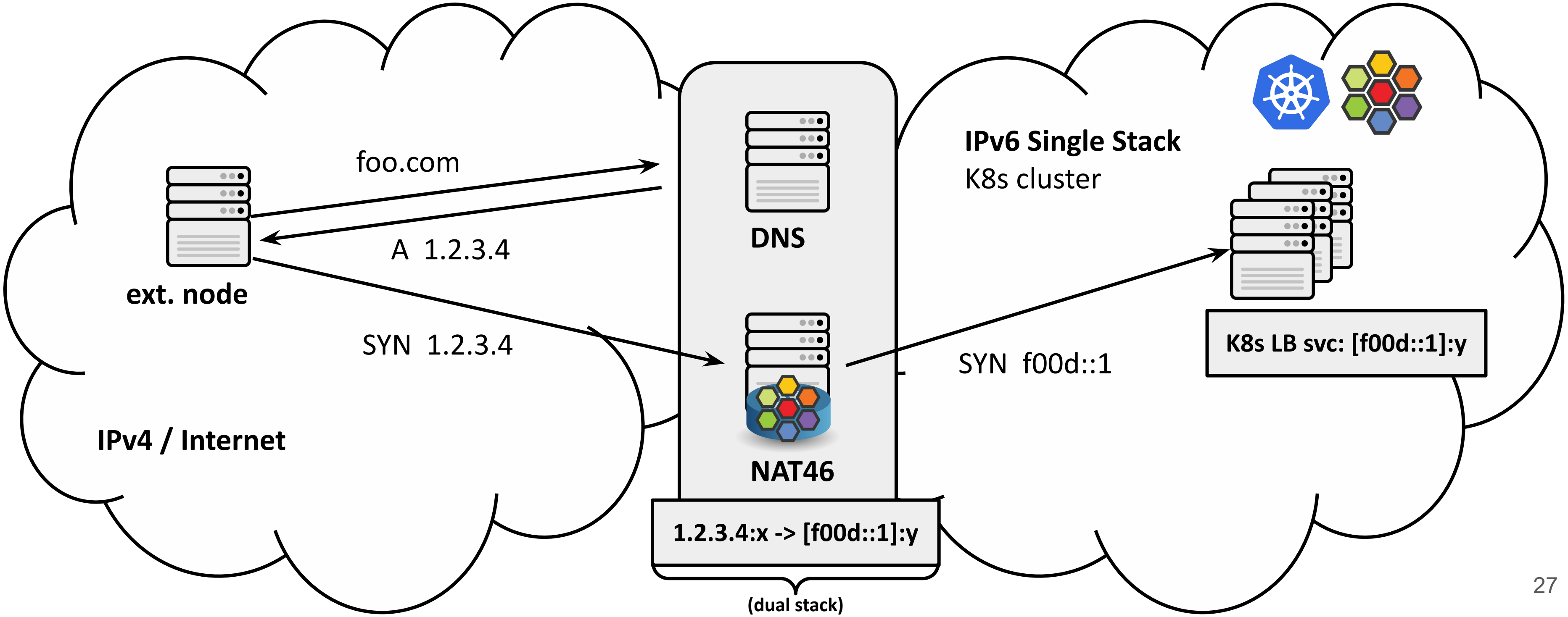
NAT46 and NAT64 with Linux? Not possible with netfilter, but eBPF can do this!

NAT46/64 eBPF support for tc BPF layer: https://lore.kernel.org/netdev/cover.1467106474.git.daniel@iogearbox.net/

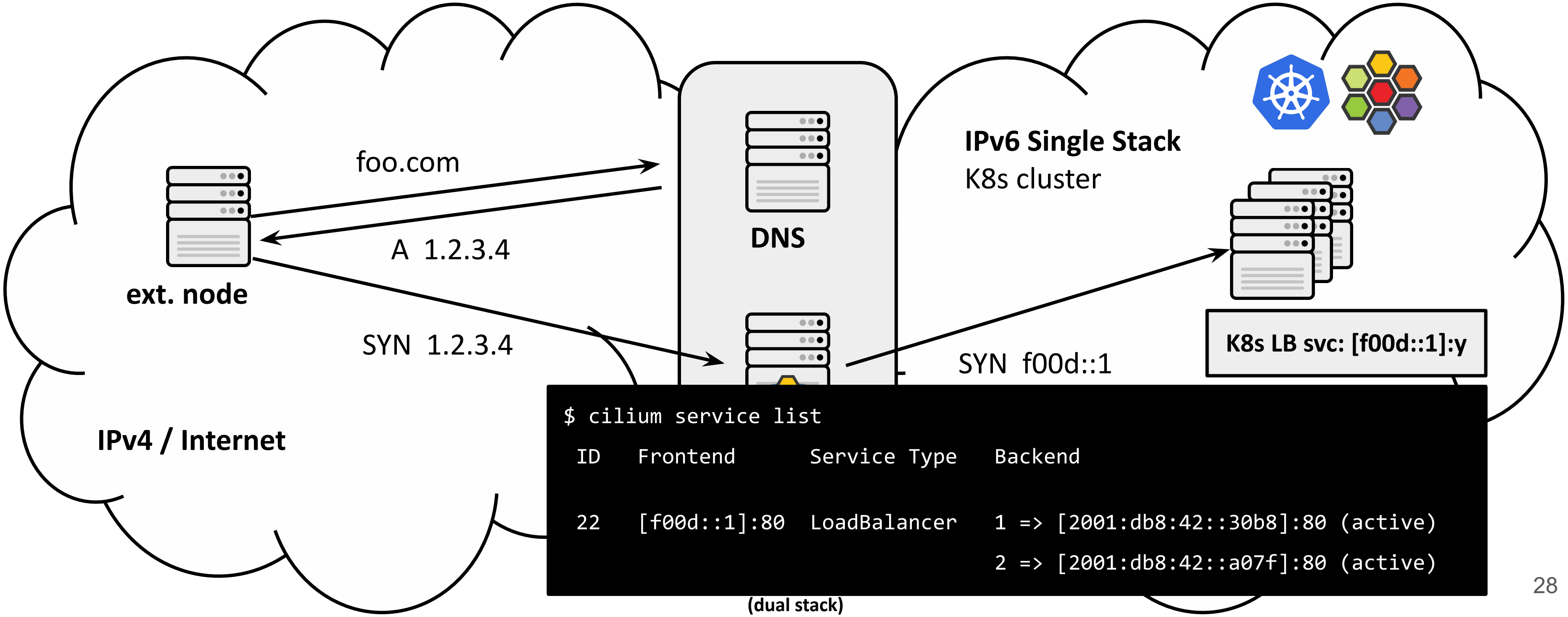
1) Ingressing IPv4 into the IPv6-only cluster

Option 1: Cilium with Stateful NAT46 GW

K8s, IPv6 and Cilium: Stateful NAT46



K8s, IPv6 and Cilium: Stateful NAT46




```
$ cilium service list
 ID   Frontend        Service Type    Backend

 22   [f00d::1]:80    LoadBalancer    1 => [2001:db8:42::30b8]:80 (active)
                                      2 => [2001:db8:42::a07f]:80 (active)
```

K8s, IPv6 and Cilium: Stateful NAT46

Standalone stateful NAT46 GW sitting at the edge and is the only Dual Stack component. K8s cluster clean IPv6-only for node/Pod IPs.

Cilium 1.12 release blog: https://isovalent.com/blog/post/cilium-release-112/#nat46-nat64

K8s, IPv6 and Cilium: Stateful NAT46

NAT46 GW mapping IPv4 VIP:port to K8s IPv6 VIP:port. IPv6 VIP:port exposed to public natively. Only IPv4 with GW hop.

K8s, IPv6 and Cilium: Stateful NAT46

Upsides: Exposed IPv4 VIP:port is completely decoupled from K8s cluster IPv6 VIP:port.

K8s, IPv6 and Cilium: Stateful NAT46

Upsides: No special LoadBalancer service needs regarding IPAM. Any public IPv6 prefix works as-is.

K8s, IPv6 and Cilium: Stateful NAT46

Upsides: NAT46 gateway in Cilium can even perform weighted Maglev load-balancing to multiple clusters.

Maglev weights merged for Cilium 1.13: https://github.com/cilium/cilium/pull/18306

K8s, IPv6 and Cilium: Stateful NAT46

Downsides: Extra control plane to program VIP to VIP mappings via API. Original source IP preservation lost.

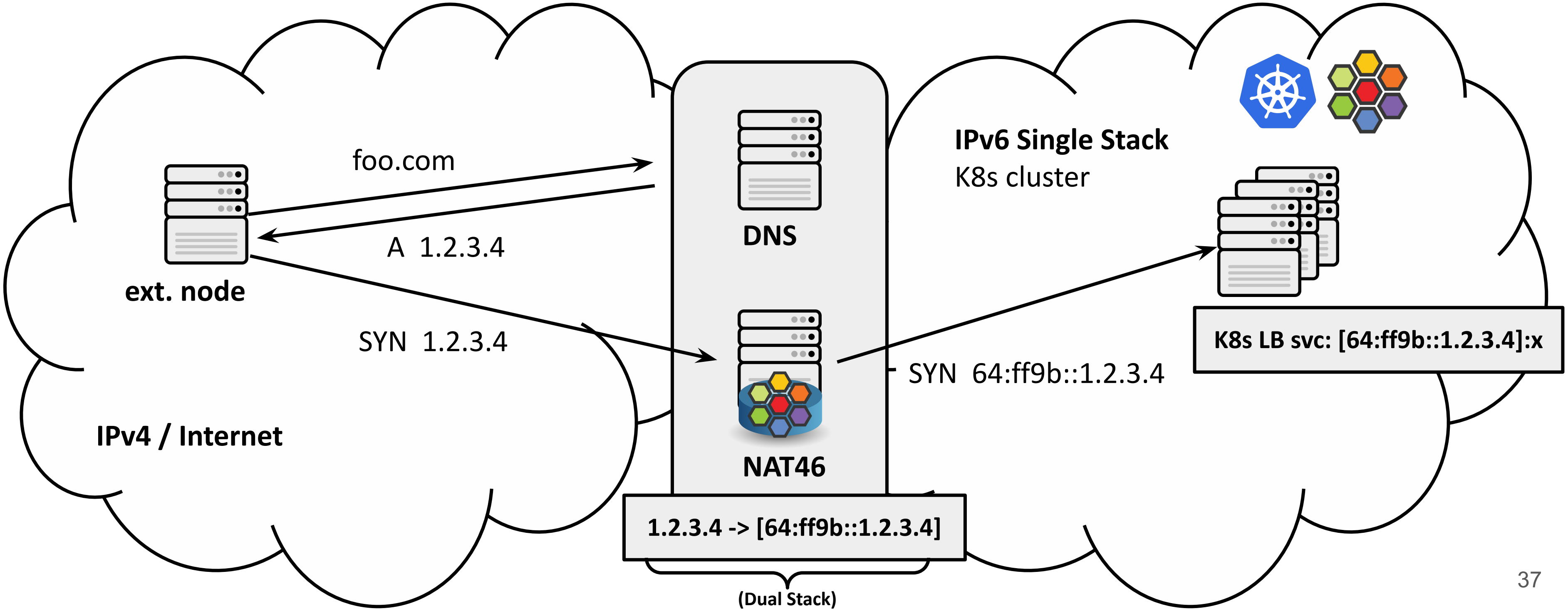
K8s, IPv6 and Cilium: Stateful NAT46

Downsides: GW stateful due to DNAT & SNAT combination, but able to absorb high packet rates thanks to eBPF & XDP.

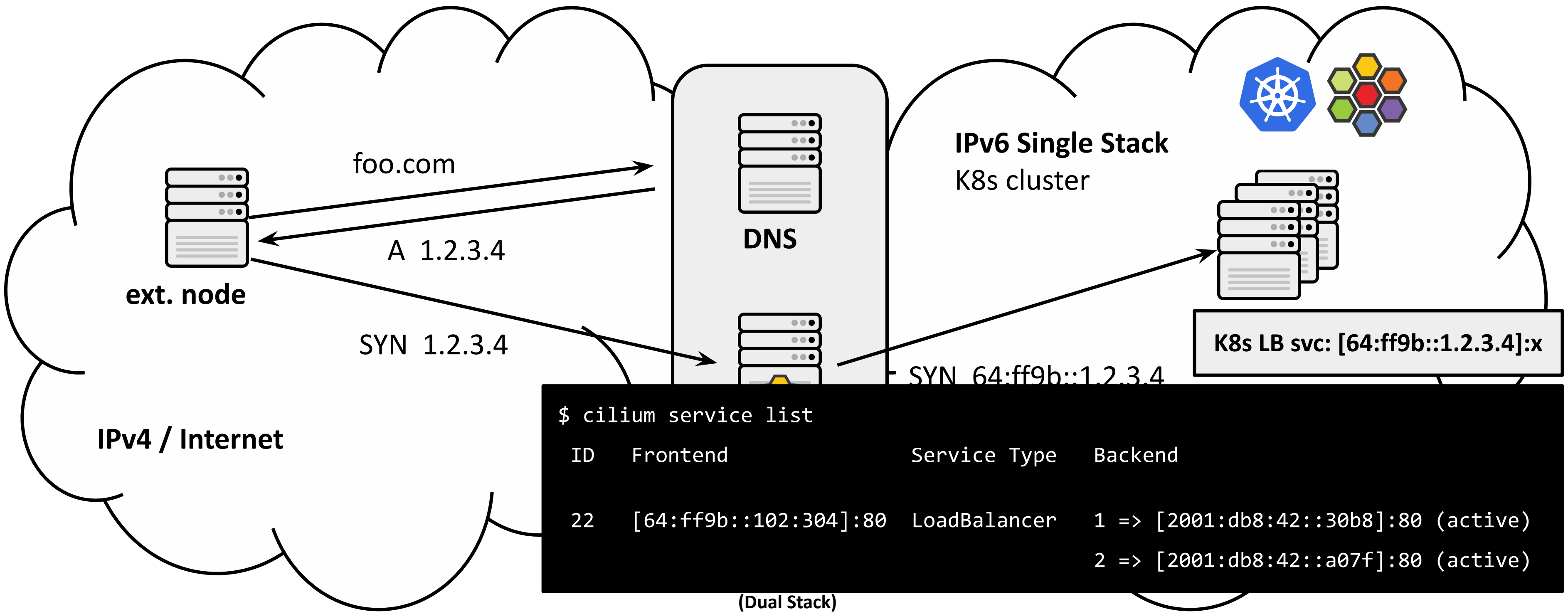
eXpress Data Path (XDP) reference paper: https://dl.acm.org/doi/10.1145/3281411.3281443

Option 2: Cilium with Stateless NAT46 GW

K8s, IPv6 and Cilium: Stateless NAT46



K8s, IPv6 and Cilium: Stateless NAT46




```
$ cilium service list
 ID    Frontend                 Service Type    Backend

 22    [64:ff9b::102:304]:80    LoadBalancer    1 => [2001:db8:42::30b8]:80 (active)
                                                2 => [2001:db8:42::a07f]:80 (active)
```

K8s, IPv6 and Cilium: Stateless NAT46

Upsides: Highly scalable GW as translation does not even hold state on the node.

K8s, IPv6 and Cilium: Stateless NAT46

Upsides: Allows for source address preservation as well given original source mapped via 64:ff9b::/96.

RFC6052: https://www.rfc-editor.org/rfc/rfc6052#section-2.1

K8s, IPv6 and Cilium: Stateless NAT46

Upsides: loadBalancerSourceRanges can then restrict LB service access for external IPv4 clients.

K8s documentation on LB source ranges (also supported on Cilium): https://kubernetes.io/docs/concepts/services-networking/_print/#aws-nlb-support

K8s, IPv6 and Cilium: Stateless NAT46

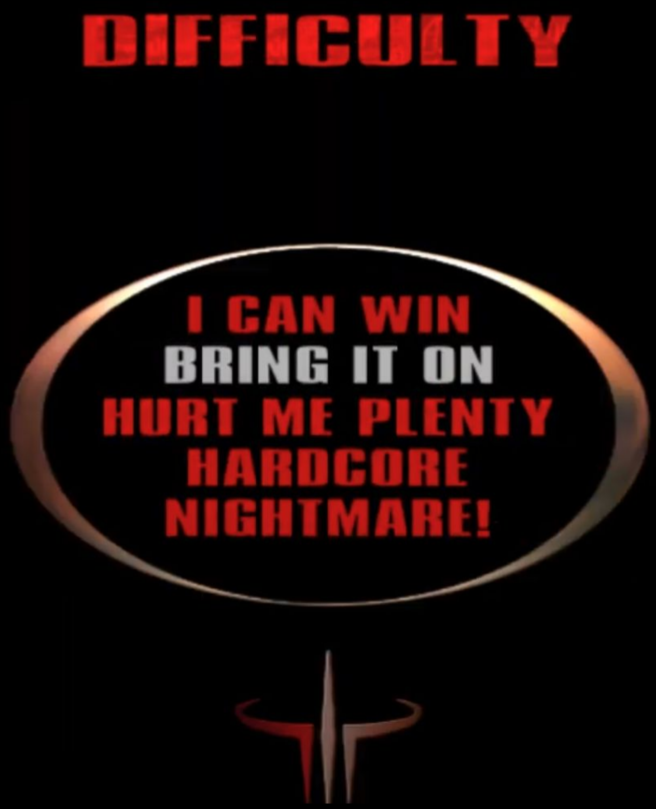
Upsides: No control plane needed given GW translation is transparent.

K8s, IPv6 and Cilium: Stateless NAT46

Downsides: LB IPAM pool in K8s cluster needs to cooperate with 64:ff9b::/96 prefix.

K8s, IPv6 and Cilium: Stateless NAT46

Downsides: Must have awareness of IPv4 mapping aspects, such that the reverse translation on GW turns into a public address.


DIFFICULTY
I CAN WIN
BRING IT ON
HURT ME PLENTY
HARDCORE
NIGHTMARE!

K8s, IPv6 and Cilium

2) Egressing from IPv6-only cluster to IPv4

K8s, IPv6 and Cilium: Stateless NAT64 + DNS64

DNS64 plays a key part ...

K8s, IPv6 and Cilium: Stateless NAT64 + DNS64

```
root@zh-lab-node-1:~# nslookup github.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   github.com
Address: 140.82.121.3
```

K8s, IPv6 and Cilium: Stateless NAT64 + DNS64

```
root@zh-lab-node-1:~# nslookup -query=AAAA github.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
*** Can't find github.com: No answer
```

K8s, IPv6 and Cilium: Stateless NAT64 + DNS64

```
root@zh-lab-node-1:~# nslookup -query=AAAA github.com 2001:4860:4860::6464
Server:         2001:4860:4860::6464
Address:        2001:4860:4860::6464#53

Non-authoritative answer:
Name:   github.com
Address: 64:ff9b::8c52:7904
```

Google's public DNS64: https://developers.google.com/speed/public-dns/docs/dns64

K8s, IPv6 and Cilium: Stateless NAT64 + DNS64

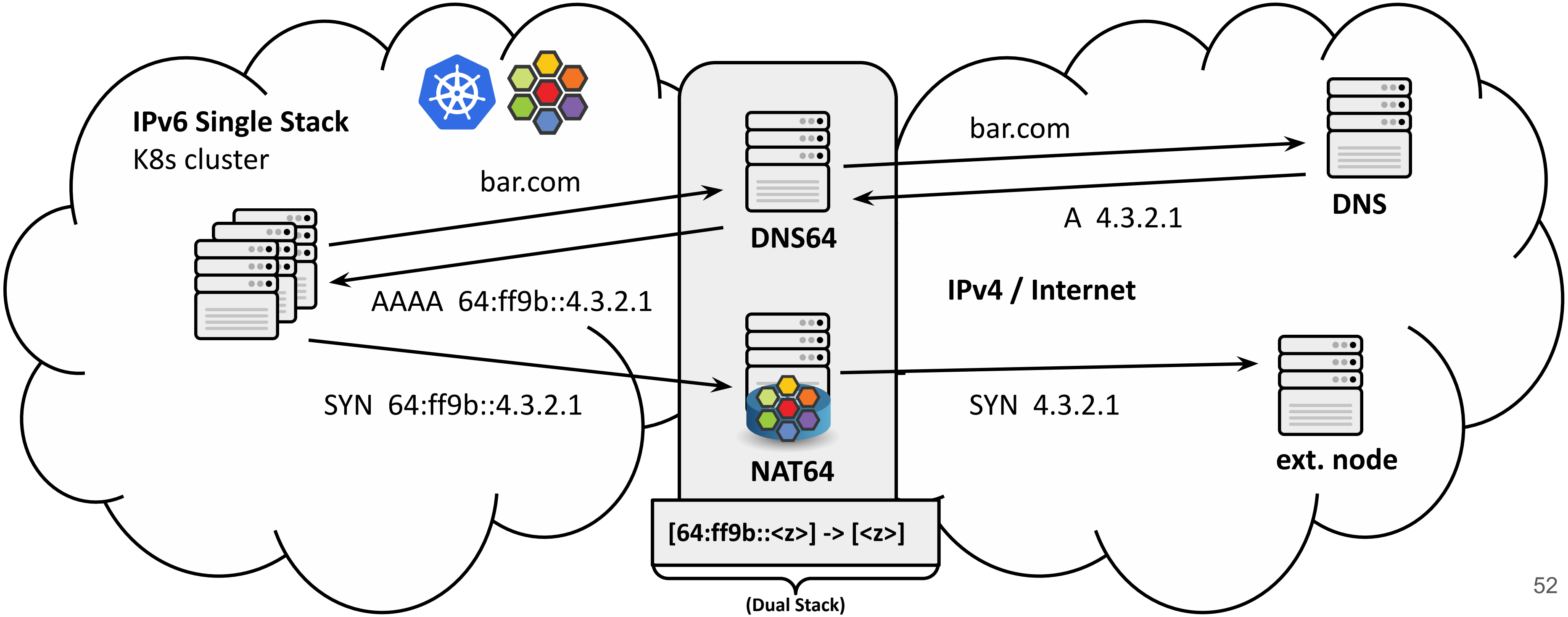
```
root@zh-lab-node-1:~# nslookup -query=AAAA github.com 2001:4860:4860::6464
Server:         2001:4860:4860::6464
Address:        2001:4860:4860::6464#53

Non-authoritative answer:
Name:   github.com
Address: 64:ff9b::8c52:7904
```

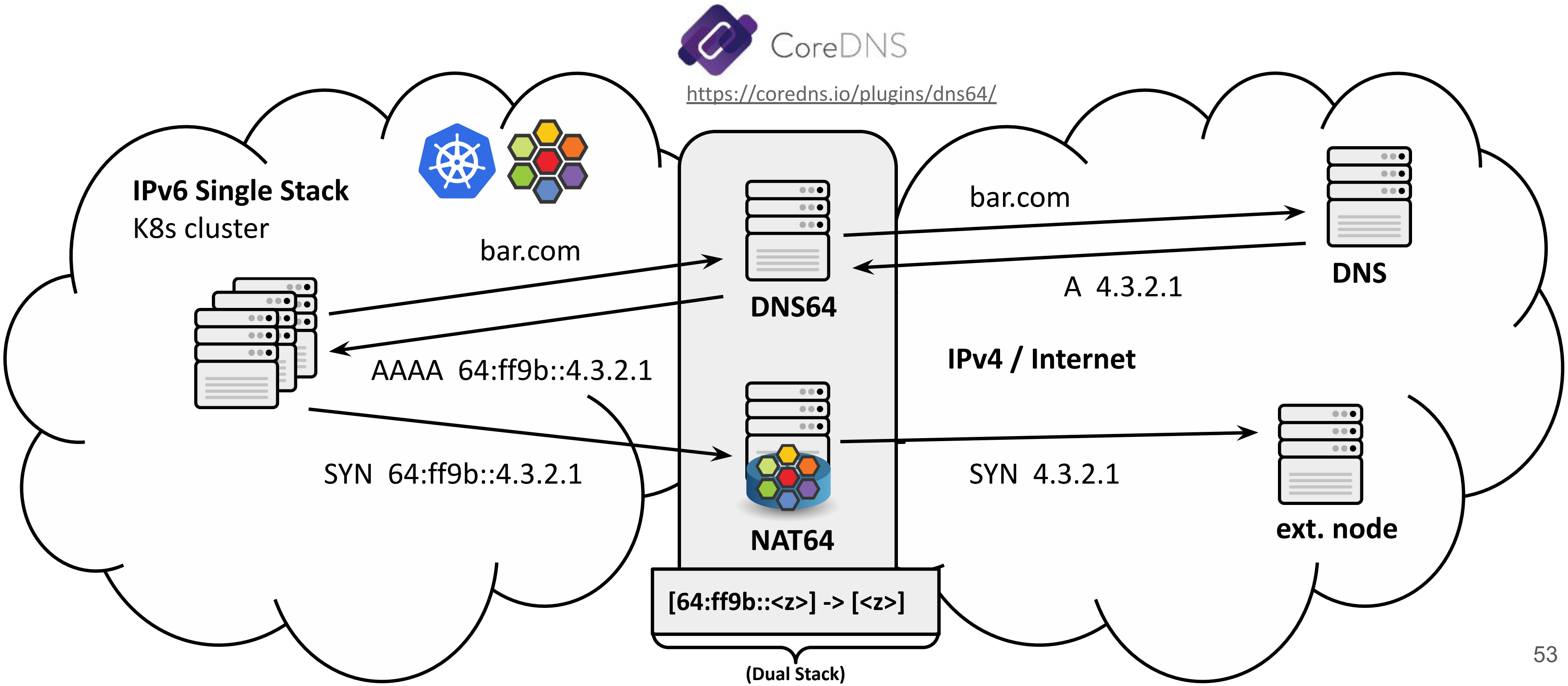
140.82.121.4

(Embedded IPv4 address)

K8s, IPv6 and Cilium: Stateless NAT64 + DNS64



K8s, IPv6 and Cilium: Stateless NAT64 + DNS64



K8s, IPv6 and Cilium: Stateless NAT64

Upsides: Again, highly scalable GW as translation does not hold state on the node.

K8s, IPv6 and Cilium: Stateless NAT64

Upsides: All traffic in the cluster between nodes/Pods/GW is pure IPv6-only.

K8s, IPv6 and Cilium: Stateless NAT64

Downsides: IPAM management more complex, since Pods/nodes need secondary 64:ff9b::/96-prefixed address.

K8s, IPv6 and Cilium: Stateless NAT64

This can be overcome via stateful NAT64 on the GW node: Pods use their primary IPv6 address, and GW does NAT to its own IPv4 address.

Demo: Cilium’s eBPF NAT46/64 GW

Not Secure | [2600:1f14:823:8a00:368f:f206:d560:1777]

Hello KubeCon !!

You've reached an IPv6-only page.

Not Secure | 52.24.201.146

Hello KubeCon !!

You've reached an IPv4-only page.

K8s and IPv6

IPv6-only cluster has been bootstrapped, can talk to IPv4. What’s next?

K8s and IPv6

IPv6 not only addresses scaling concerns,
but also future performance requirements.
Enter: Cilium with BIG TCP

IPv6, Cilium: BIG TCP

BIG TCP got merged into v5.19 Linux kernels.

Goal: Support data center workloads for single socket towards 100Gbit/s and beyond.

Going big with TCP packets: https://lwn.net/Articles/884104/

tcp: BIG TCP implementation: https://lore.kernel.org/netdev/20220513183408.686447-1-eric.dumazet@gmail.com/

IPv6, Cilium: BIG TCP

Why care? Big data, AI/ML and other network-intense workloads but also more generally to free up resources for apps.

Going big with TCP packets: https://lwn.net/Articles/884104/

tcp: BIG TCP implementation: https://lore.kernel.org/netdev/20220513183408.686447-1-eric.dumazet@gmail.com/

IPv6, Cilium: BIG TCP

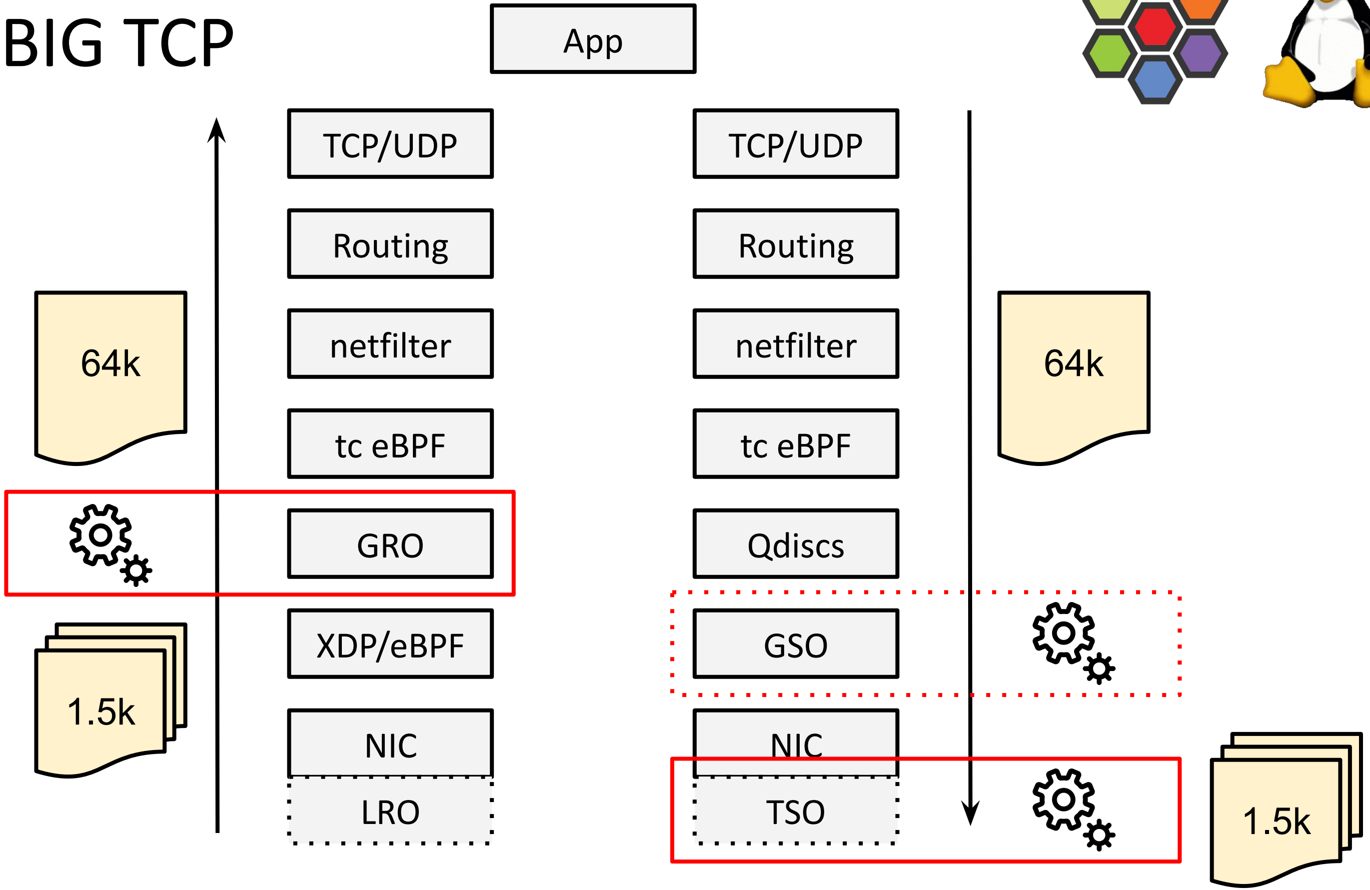
100Gbit/s == 123ns per packet (8.15Mpps)
with 1.5k MTU for single socket sink.

IPv6, Cilium: BIG TCP

Unrealistic for upper kernel stack.

Batching is needed: GRO/TSO

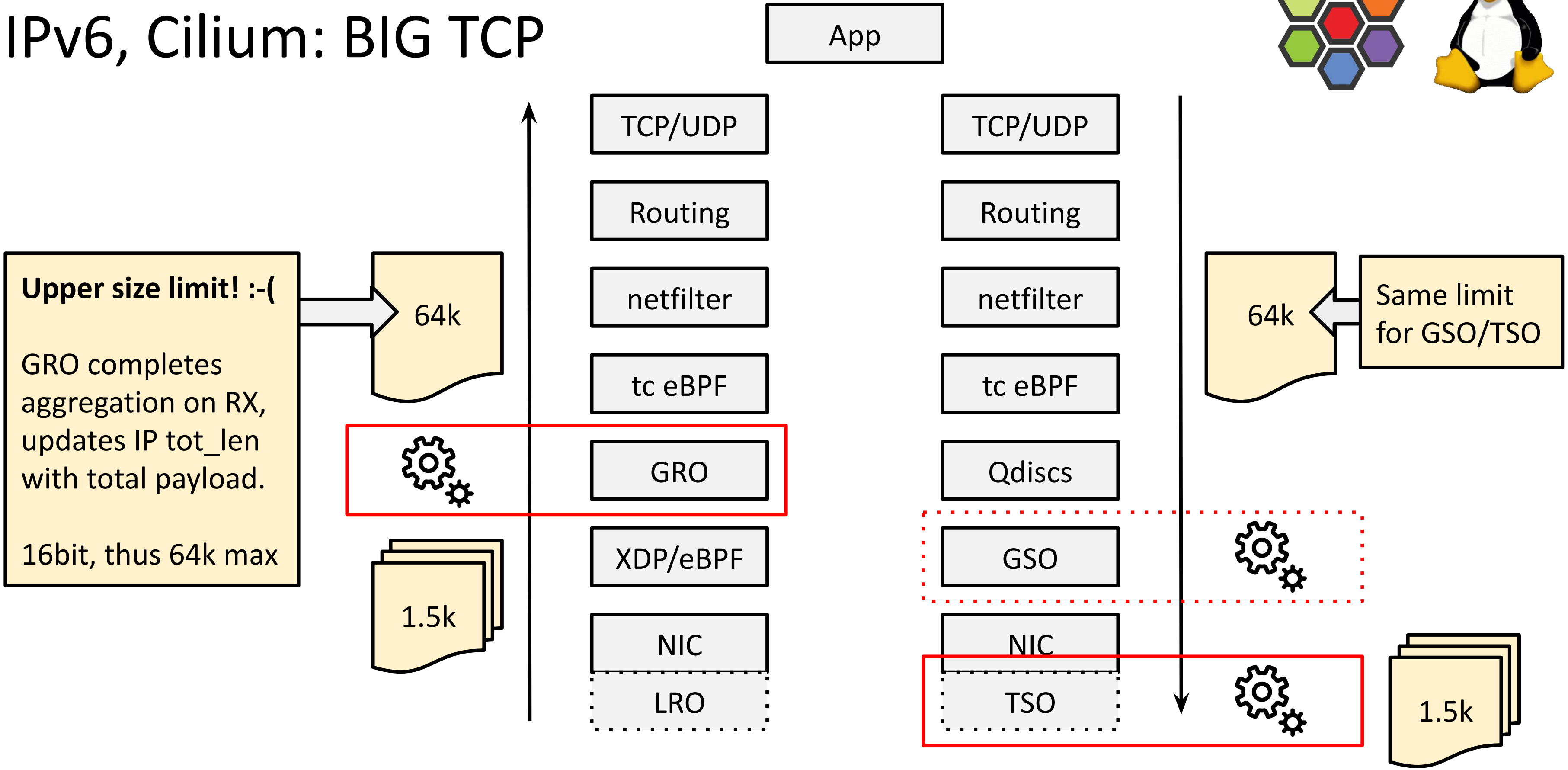
IPv6, Cilium: BIG TCP



IPv6, Cilium: BIG TCP

TSO segments TCP super-sized packet in NIC/HW, and GRO on receiver gets packet train, reconstructs super-sized packet.

IPv6, Cilium: BIG TCP



IPv6, Cilium: BIG TCP

Can kernel make bigger batches?

IPv6 to the rescue.

IPv6, Cilium: BIG TCP

BIG TCP overcomes this 16bit limit with relatively small changes by locally inserting a Hop-By-Hop (HBH) IPv6 extension header.

Going big with TCP packets: https://lwn.net/Articles/884104/

tcp: BIG TCP implementation: https://lore.kernel.org/netdev/20220513183408.686447-1-eric.dumazet@gmail.com/

IPv6, Cilium: BIG TCP

locally == node local, meaning the HBH header does not go onto wire. Also no MTU tweaks required.

Going big with TCP packets: https://lwn.net/Articles/884104/

tcp: BIG TCP implementation: https://lore.kernel.org/netdev/20220513183408.686447-1-eric.dumazet@gmail.com/

IPv6, Cilium: BIG TCP

Ethernet header

IPv6 header → 16 bit ‘only’ payload_len field must be zero

Hop-by-Hop with TLV_JUMBO (8b) → payload_len field is 32bit here

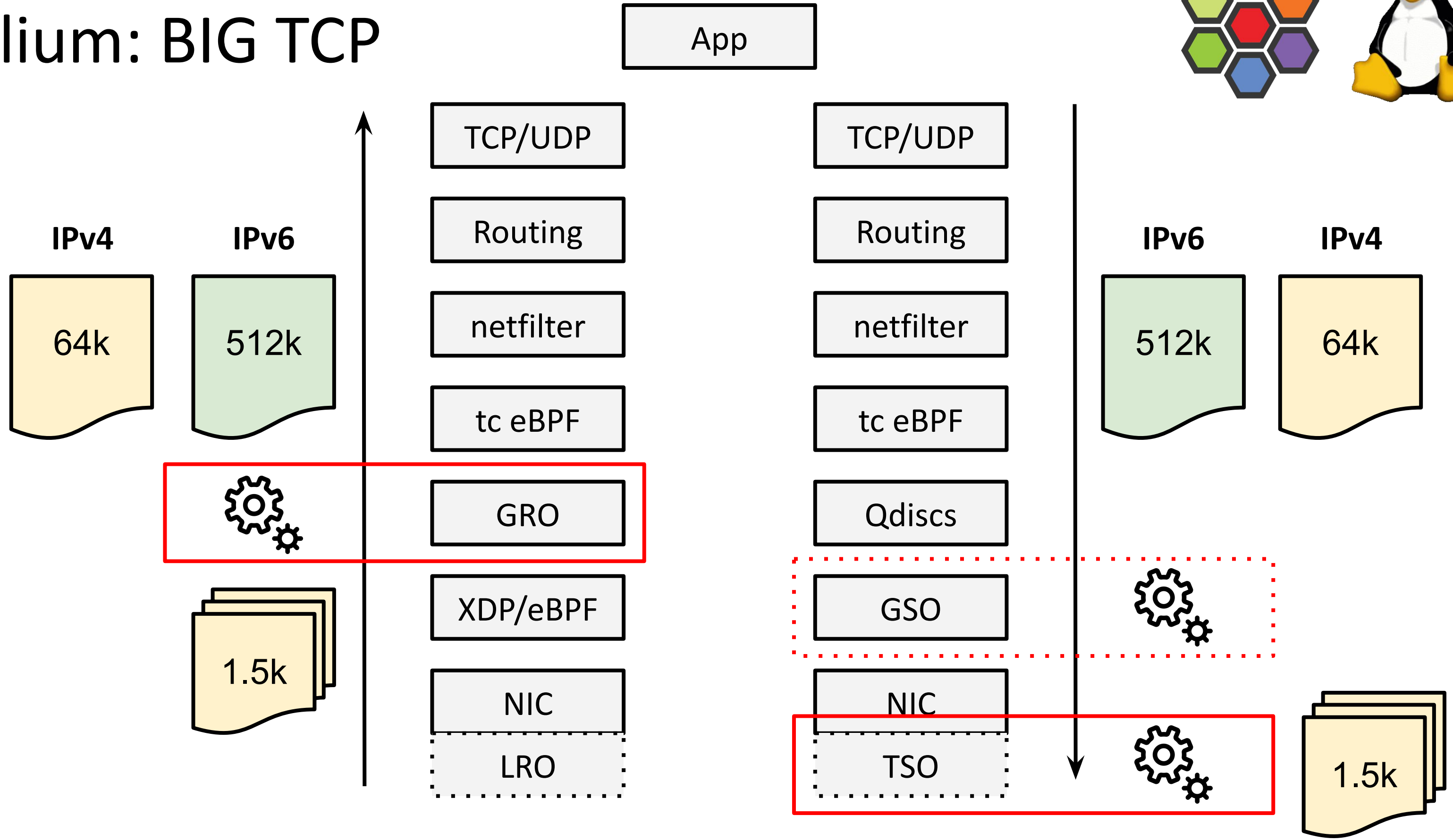
TCP header

Payload (aggregated)

Going big with TCP packets: https://lwn.net/Articles/884104/

tcp: BIG TCP implementation: https://lore.kernel.org/netdev/20220513183408.686447-1-eric.dumazet@gmail.com/

IPv6, Cilium: BIG TCP



IPv6, Cilium: BIG TCP

Upper aggregation limit is 512k, but this can be easily raised further in future when needed. IPv4 stuck with 64k limit.

IPv6, Cilium: BIG TCP

Upcoming Cilium 1.13:

```
helm install cilium cilium/cilium \
  [...]
  --set enableIPv6BIGTCP=true
```

Cilium 1.13's BIG TCP docs: https://docs.cilium.io/en/latest/operations/performance/tuning/#ipv6-big-tcp

Cilium 1.13's BIG TCP implementation: https://github.com/cilium/cilium/pull/20349

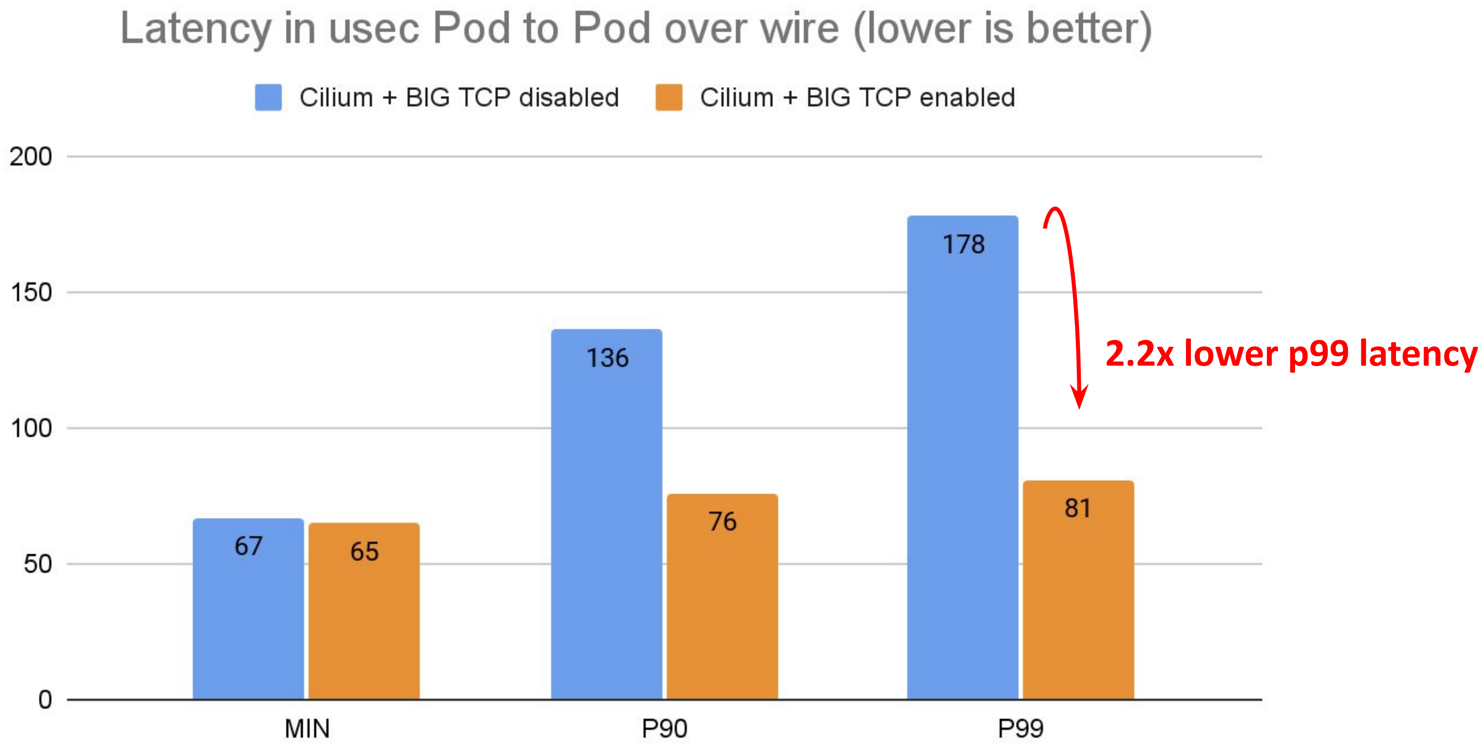
IPv6, Cilium: BIG TCP

Cilium then sets up IPv6 BIG TCP for all host and Pod devices transparently. This also helps request/response-type workloads!

Cilium 1.13's BIG TCP docs: https://docs.cilium.io/en/latest/operations/performance/tuning/#ipv6-big-tcp

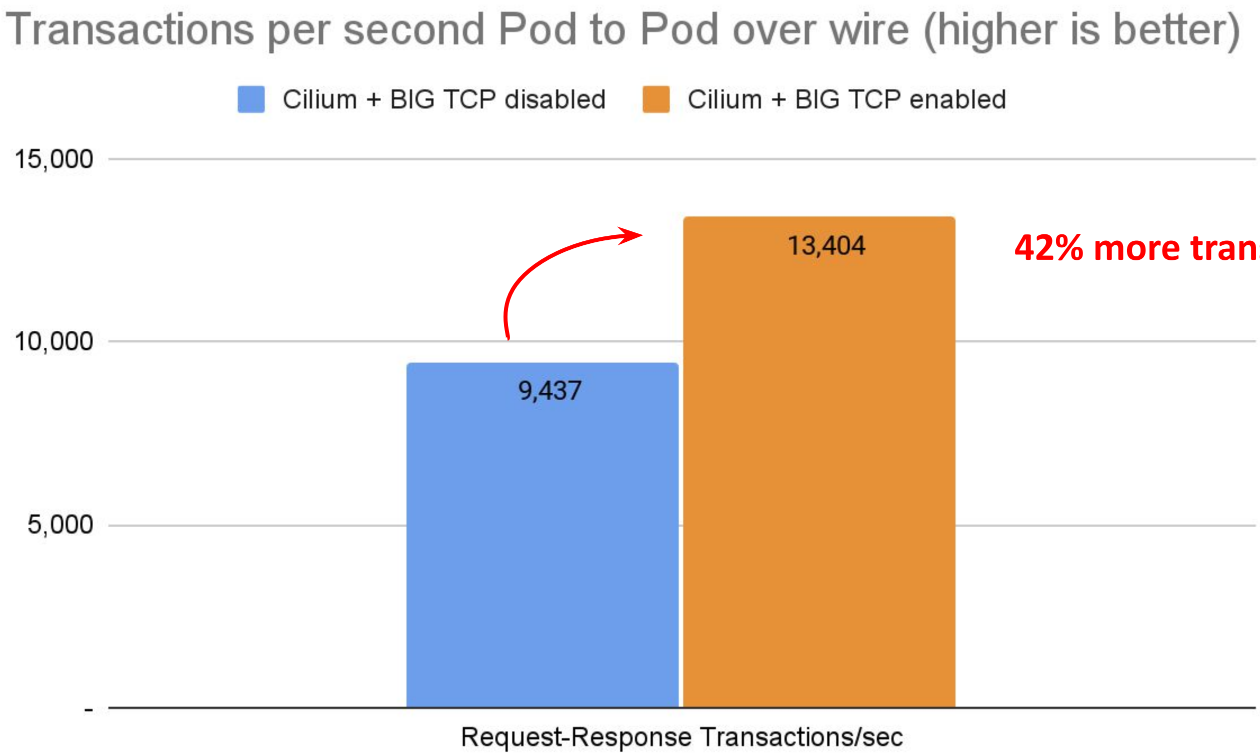
Cilium 1.13's BIG TCP implementation: https://github.com/cilium/cilium/pull/20349

IPv6, Cilium: BIG TCP



Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver
netperf -t TCP_RR -H <remote pod> -- -r 80000,80000 -O MIN_LATENCY,P90_LATENCY,P99_LATENCY,THROUGHPUT

IPv6, Cilium: BIG TCP



42% more transactions/sec

Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver
netperf -t TCP_RR -H <remote pod> -- -r 80000,80000 -O MIN_LATENCY,P90_LATENCY,P99_LATENCY,THROUGHPUT

IPv6, Cilium: BIG TCP

```
root@zh-lab-node-3:~# ip netns exec test-veth iperf3 -c fd00::1 -t 55 -i 5 -O 5
Connecting to host fd00::1, port 5201
[  5] local fd00::2 port 34972 connected to fd00::1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-5.00   sec  32.1 GBytes  55.2 Gbits/sec  18120   2.60 MBytes       (omitted)
[  5]   0.00-5.00   sec  31.2 GBytes  53.6 Gbits/sec  9460   3.08 MBytes
[  5]   5.00-10.00  sec  30.1 GBytes  51.7 Gbits/sec  12393   2.61 MBytes
[  5]  10.00-15.00  sec  29.5 GBytes  50.7 Gbits/sec  14120   2.70 MBytes
[  5]  15.00-20.00  sec  30.7 GBytes  52.7 Gbits/sec  10659   2.87 MBytes
[  5]  20.00-25.00  sec  29.8 GBytes  51.2 Gbits/sec  8889   2.81 MBytes
[  5]  25.00-30.00  sec  30.1 GBytes  51.7 Gbits/sec  9806   2.36 MBytes
[  5]  30.00-35.00  sec  31.0 GBytes  53.3 Gbits/sec  9452   2.65 MBytes
[  5]  35.00-40.00  sec  30.8 GBytes  52.9 Gbits/sec  8471   2.67 MBytes
[  5]  40.00-45.00  sec  30.1 GBytes  51.8 Gbits/sec  9662   2.07 MBytes
[  5]  45.00-50.00  sec  30.3 GBytes  52.1 Gbits/sec  10148   1.90 MBytes
[  5]  50.00-55.00  sec  30.2 GBytes  51.9 Gbits/sec  9924   2.37 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-55.00  sec   334 GBytes  52.1 Gbits/sec  112984             sender
[  5]   0.00-55.00  sec   334 GBytes  52.1 Gbits/sec                  receiver
```

Default, BIG TCP off.

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

IPv6, Cilium: BIG TCP

```
root@zh-lab-node-3:~# ip netns exec test-veth iperf3 -c fd00::1 -t 55 -i 5 -O 5
Connecting to host fd00::1, port 5201
[  5] local fd00::2 port 44408 connected to fd00::1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-5.00   sec  35.4 GBytes  60.7 Gbits/sec  22825   3.04 MBytes       (omitted)
[  5]   0.00-5.00   sec  35.0 GBytes  60.2 Gbits/sec  5335   3.23 MBytes
[  5]   5.00-10.00  sec  35.1 GBytes  60.2 Gbits/sec  5961   2.87 MBytes
[  5]  10.00-15.00  sec  34.6 GBytes  59.5 Gbits/sec  6523   3.11 MBytes
[  5]  15.00-20.00  sec  34.1 GBytes  58.5 Gbits/sec  7695   2.54 MBytes
[  5]  20.00-25.00  sec  34.8 GBytes  59.8 Gbits/sec  4494   3.80 MBytes
[  5]  25.00-30.00  sec  35.6 GBytes  61.1 Gbits/sec  5468   2.30 MBytes
[  5]  30.00-35.00  sec  35.0 GBytes  60.1 Gbits/sec  4176   3.34 MBytes
[  5]  35.00-40.00  sec  36.2 GBytes  62.2 Gbits/sec  5066   4.25 MBytes
[  5]  40.00-45.00  sec  33.8 GBytes  58.1 Gbits/sec  5470   2.73 MBytes
[  5]  45.00-50.00  sec  35.3 GBytes  60.6 Gbits/sec  5181   2.82 MBytes
[  5]  50.00-55.00  sec  34.4 GBytes  59.1 Gbits/sec  6898   1.93 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-55.00  sec   384 GBytes  60.0 Gbits/sec  62267             sender
[  5]   0.00-55.01  sec   384 GBytes  59.9 Gbits/sec                  receiver
```

Updated, BIG TCP on.

+8 Gbit/s delta for app in our test setup

IPv6, Cilium: BIG TCP

```
- 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
```

(**iperf3** does not support mmap()'ed TCP. Biggest overhead in this test is copy to/from user. Here limited to ~60Gbps.)

IPv6, Cilium: BIG TCP

More broadly, BIG TCP is one piece of the puzzle of Cilium’s overall architecture.

Cilium: Tomorrow’s Data Plane

Key features of Cilium’s high-scale, opinionated dataplane (extract):

Cilium as a standalone gateway:

➔ eBPF/XDP Layer 4 Load-Balancer with programmable API
- Weighted Maglev consistent hashing
- DSR with various encaps like IPIP/IP6IP6 and backend RSS fanout
- Graceful Backend Termination/Quarantining

➔ **New:** Stateful NAT46/64 Gateway

➔ **New:** Stateless NAT46/64 Gateway
- Both enable IPv6-only K8s clusters on/off-prem

Cilium: Tomorrow's Data Plane

Key features of Cilium's high-scale, opinionated dataplane (extract):

Cilium as a networking platform in Kubernetes:

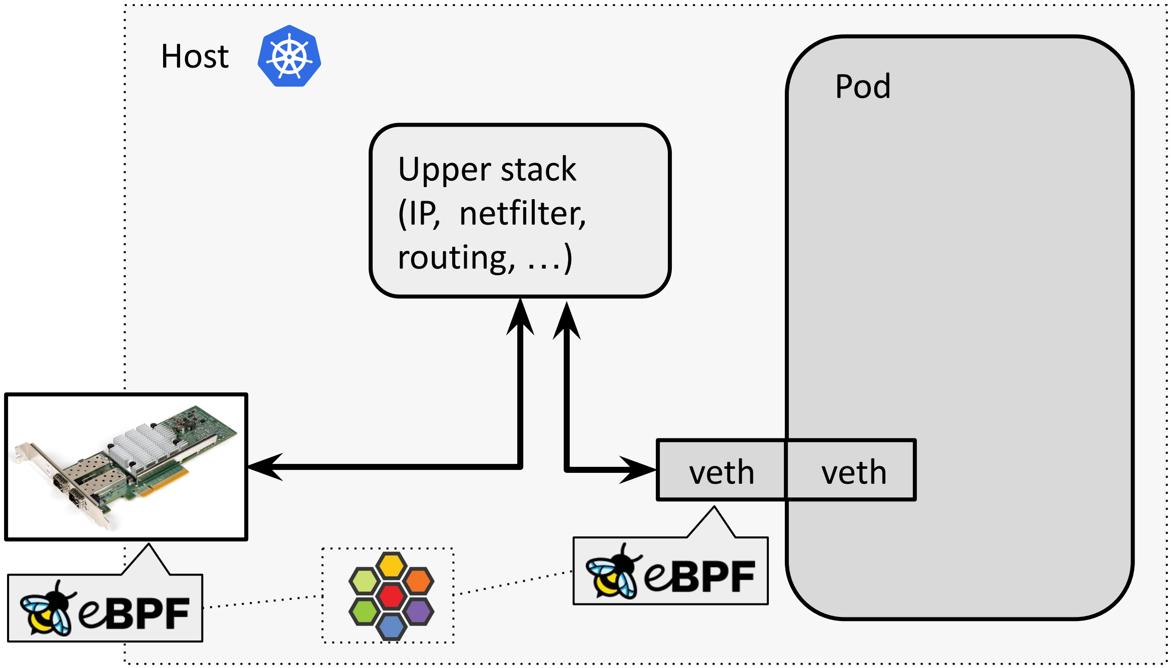
- ➔ eBPF kube-proxy replacement with XDP and socket LB support
- ➔ eBPF host routing with low-latency forwarding to Pods
- ➔ Bandwidth Manager infrastructure (KubeCon EU 2022)
  - EDT rate-limiting via eBPF and MQ/FQ
  - Pacing and BBR support for Pods
  - Disabling TCP slow start after idle
- ➔ **New:** IPv6 BIG TCP support
- ➔ **New:** eBPF meta driver for Pods as veth device replacement

Cilium: meta devices for eBPF

~~veth~~ meta netdevices ...

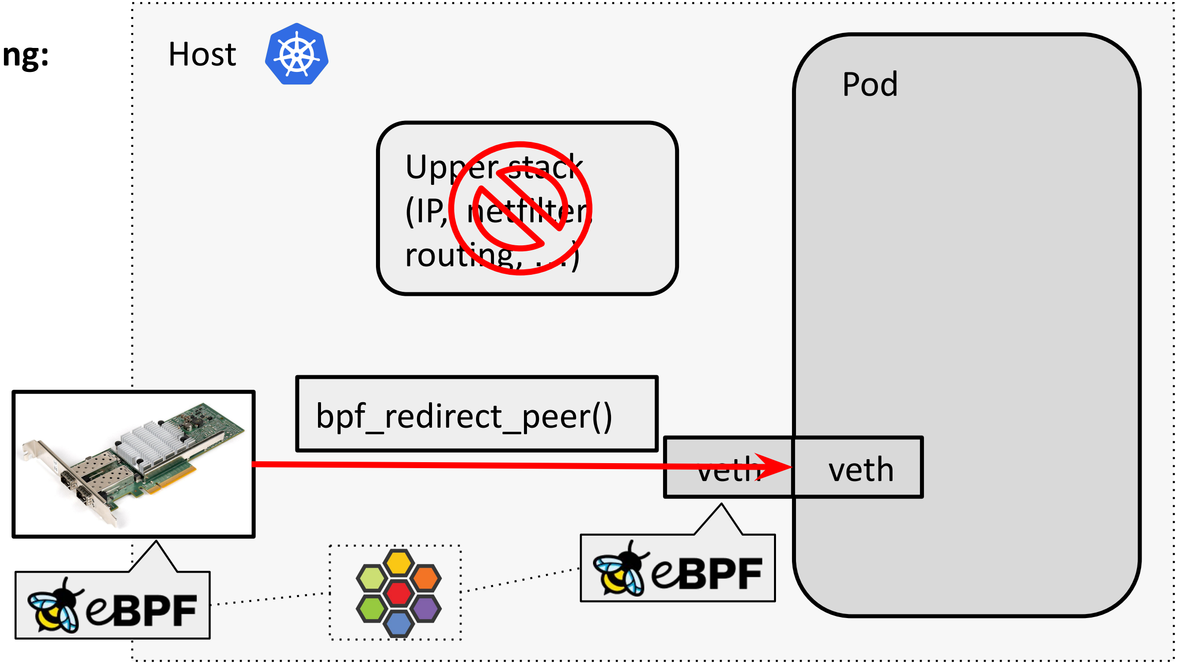
Cilium: meta devices for eBPF

Default case:



Cilium: meta devices for eBPF

eBPF host routing:

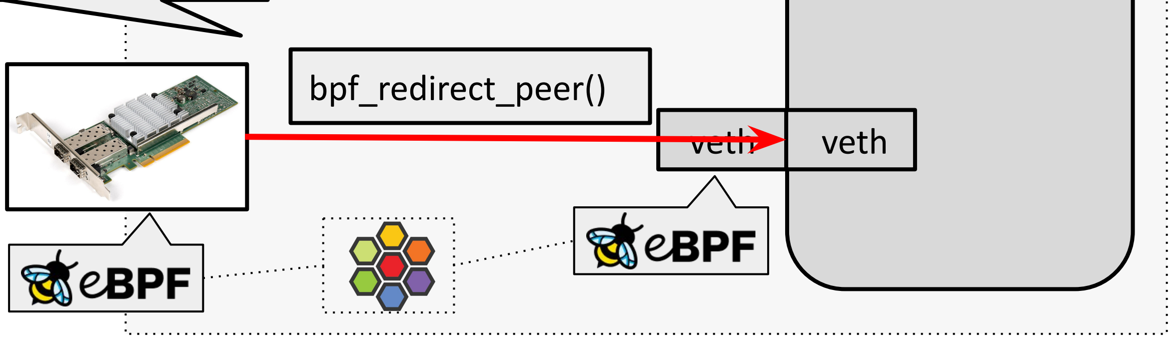


Virtual Ethernet Device Optimization with eBPF: https://cilium.io/blog/2020/11/10/cilium-19/#veth

Cilium: meta devices for eBPF

Internals:

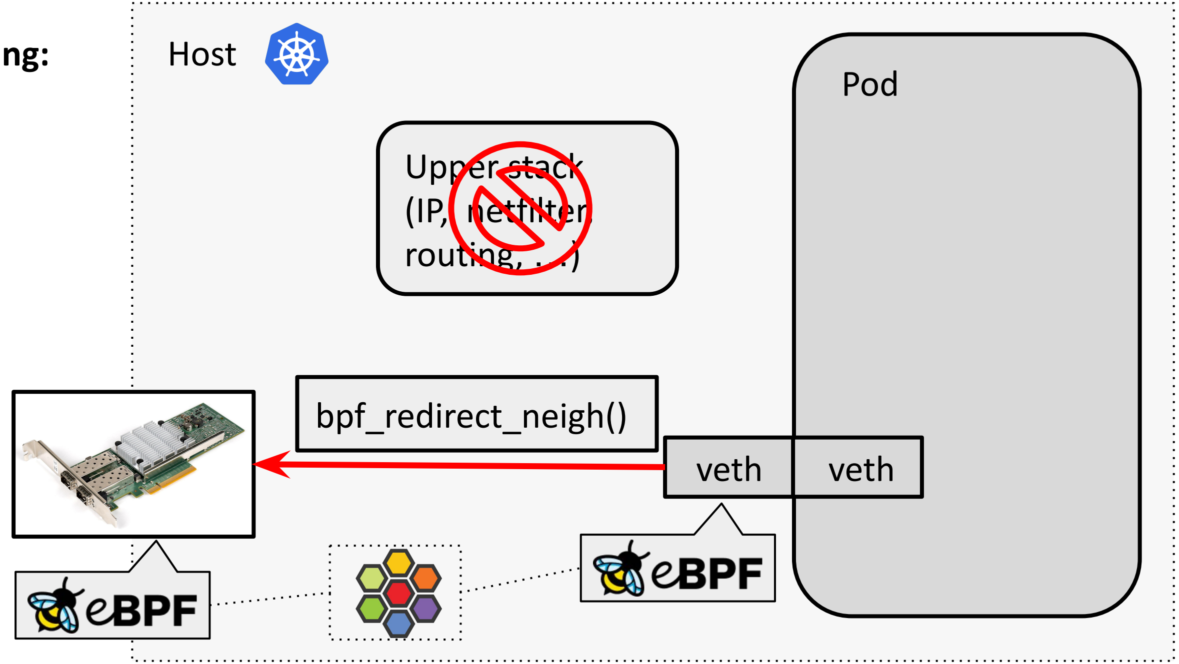
```
dev = ops->ndo_get_peer_dev(dev)
skb_scrub_packet()
skb->dev = dev
sch_handle_ingress():
 - goto another_round
 - no per-CPU backlog queue
```



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

Cilium: meta devices for eBPF

eBPF host routing:

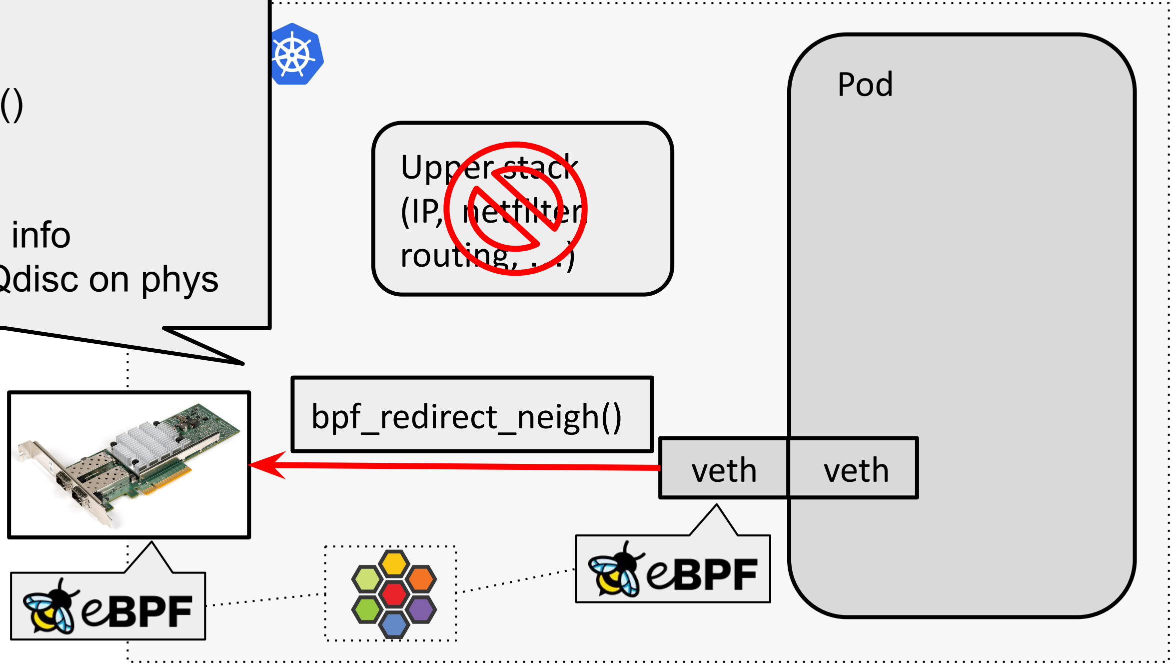


Virtual Ethernet Device Optimization with eBPF: https://cilium.io/blog/2020/11/10/cilium-19/#veth

Cilium: meta devices for eBPF

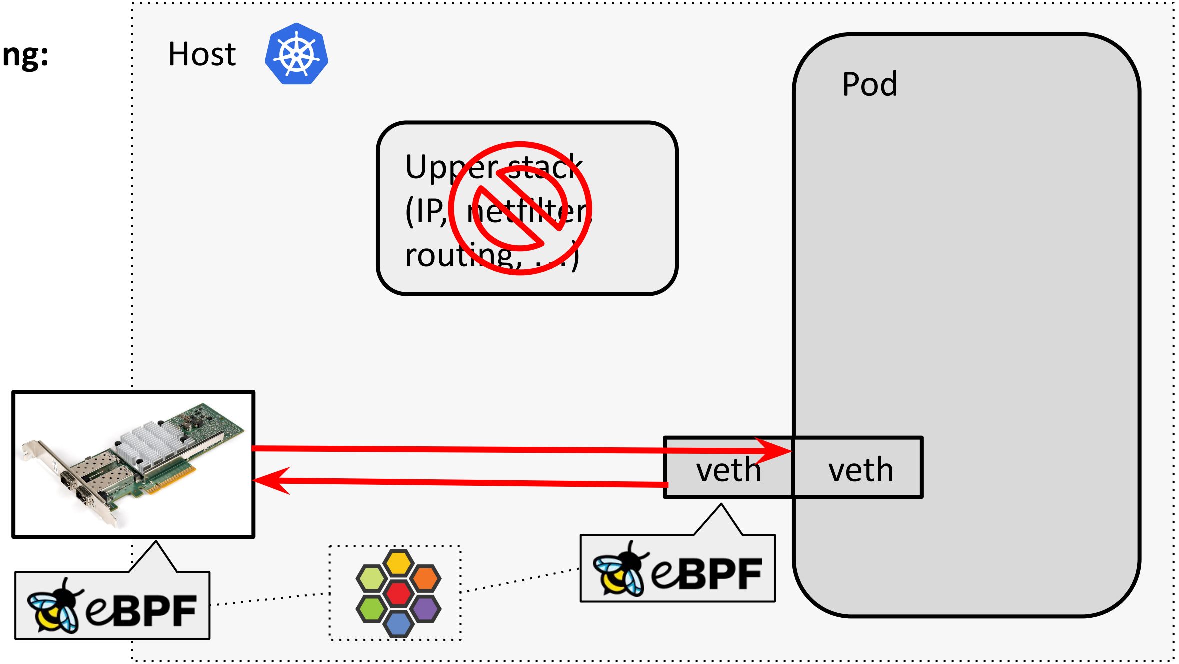
Internals:

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



Cilium: meta devices for eBPF

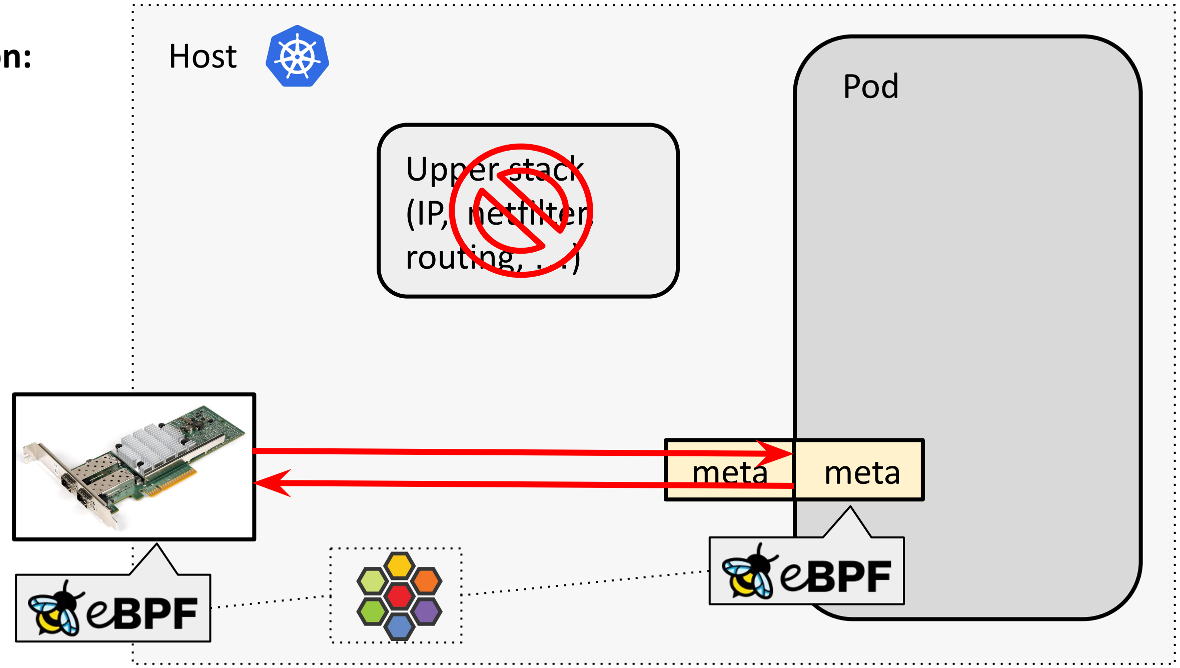
eBPF host routing:



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

Cilium: meta devices for eBPF

meta in addition:



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

Cilium: meta devices for eBPF

For meta devices the eBPF program becomes part of the device itself *inside* the Pod.

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

Cilium: meta devices for eBPF

But without the Pod being able to change or unload the eBPF program. Solely controlled from Cilium in host namespace.

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

Cilium: meta devices for eBPF

We designed meta devices specifically for Cilium to shift Pod-specific BPF programs from tc BPF into meta layer.

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

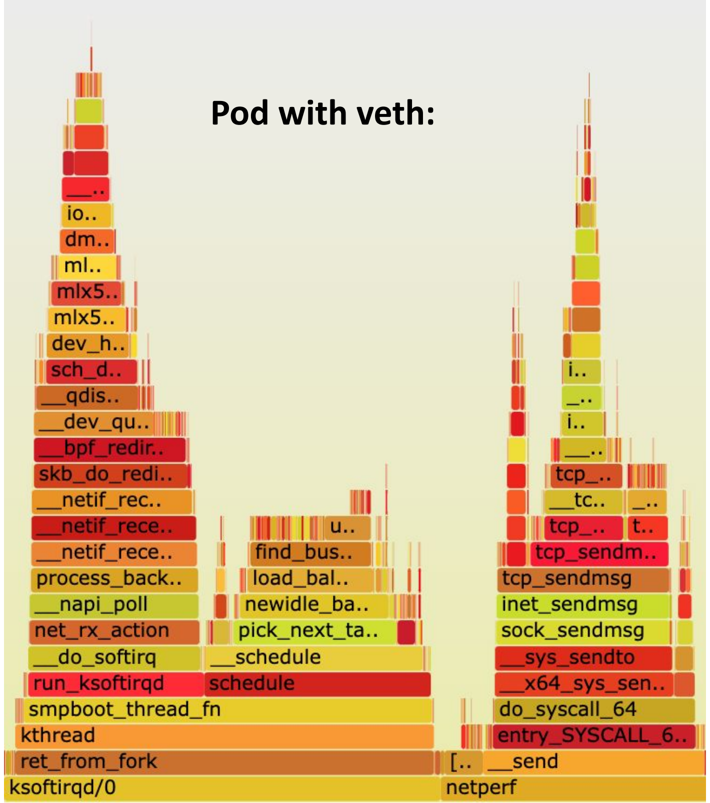
Cilium: meta devices for eBPF

This reduces latency for applications inside Pods as close to those residing in host namespace.

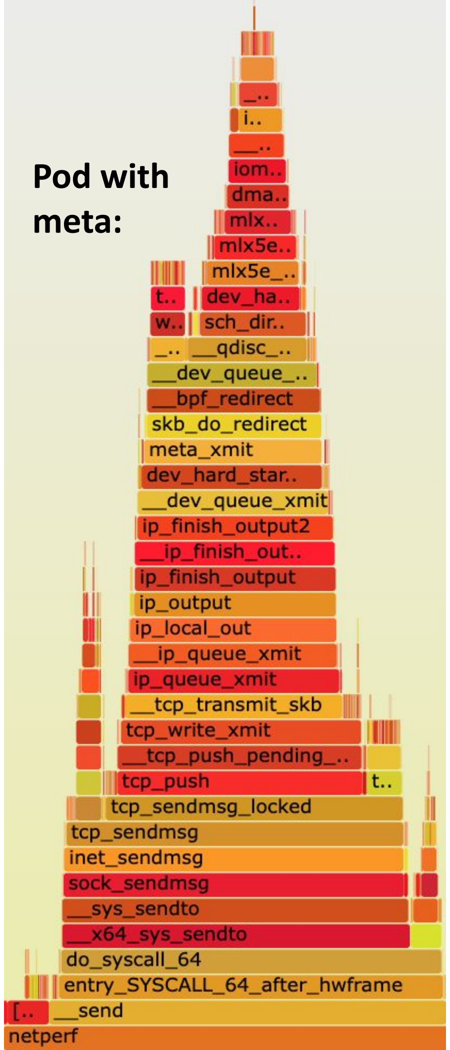
meta prototype: https://github.com/cilium/linux/commits/pr/dev-meta

Cilium: meta devices for eBPF

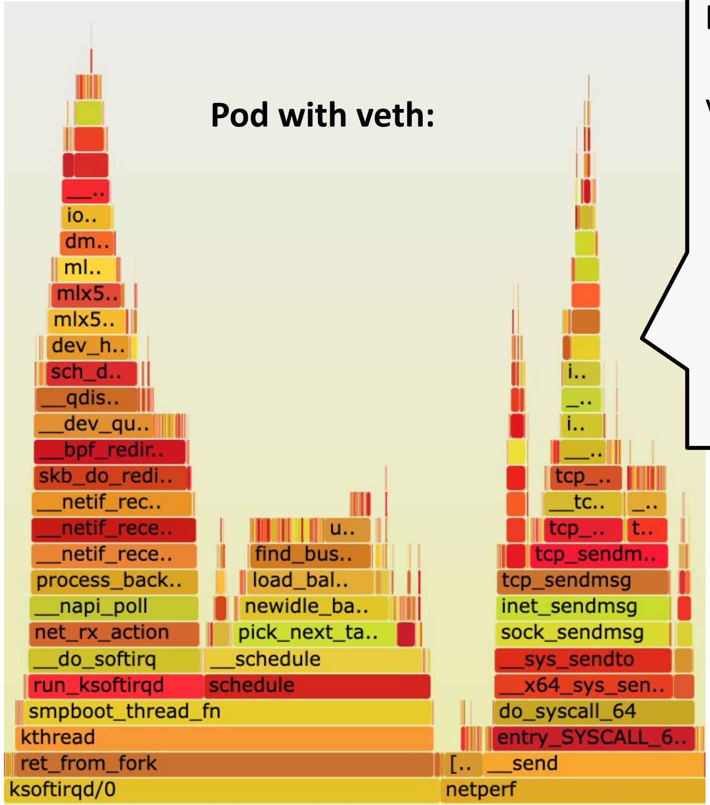
Pod with veth:

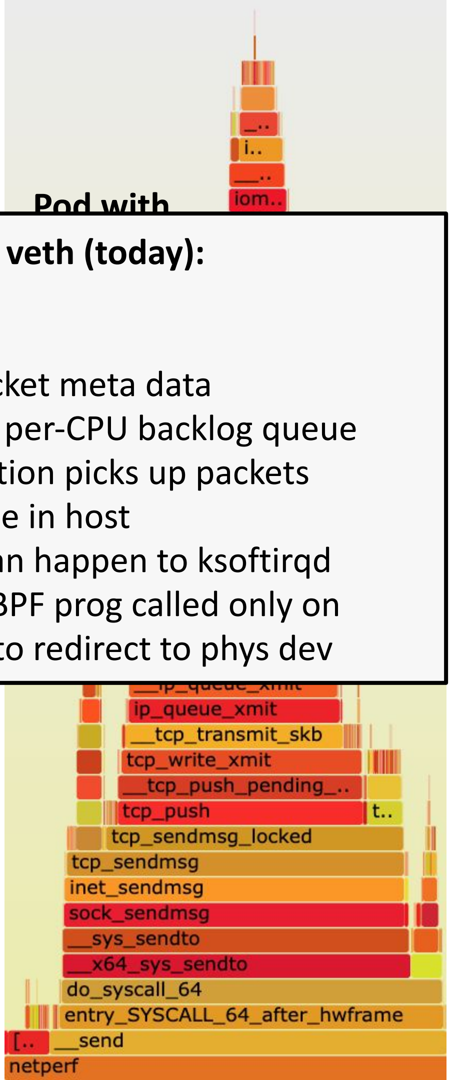


Pod with meta:



Cilium: meta devices for eBPF

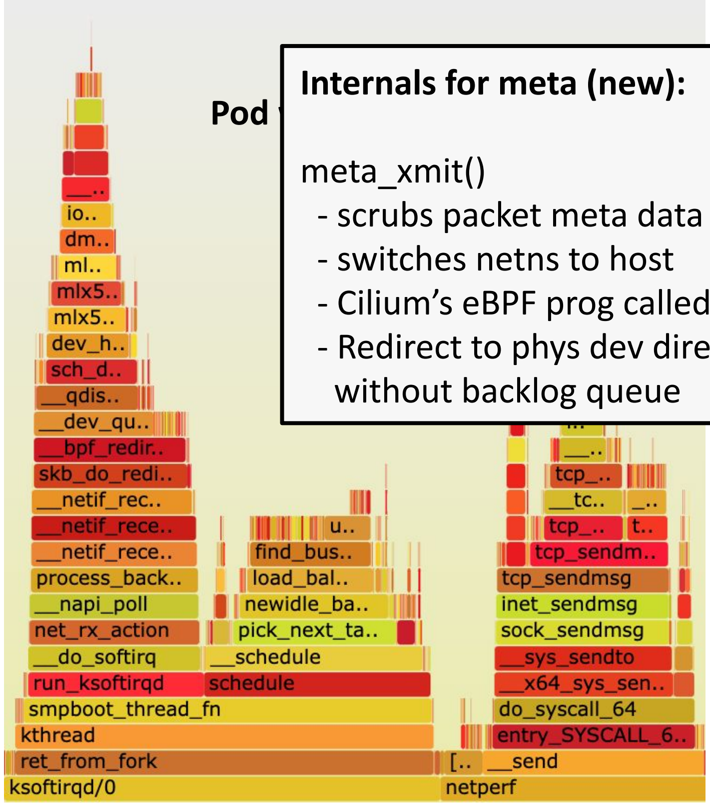




Internals for veth (today):

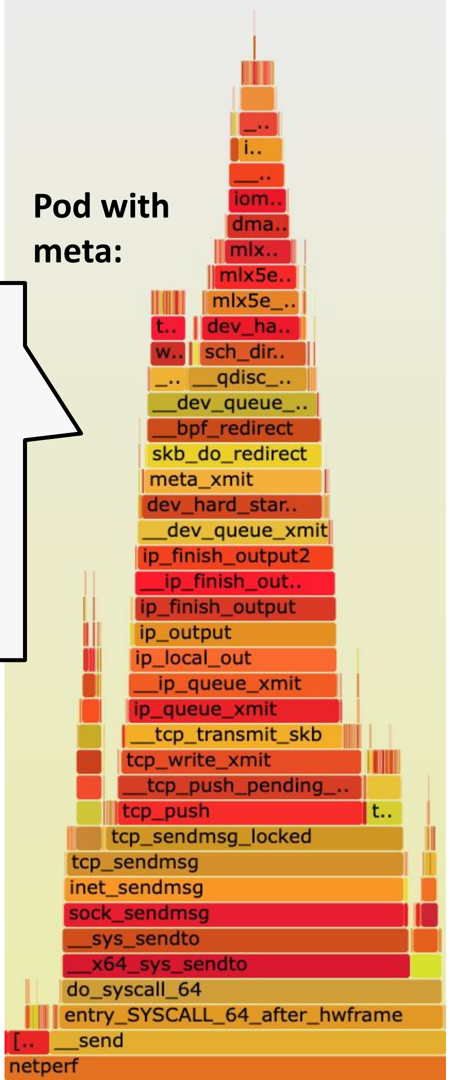
veth_xmit()
- scrubs packet meta data
- enques to per-CPU backlog queue
- net_rx_action picks up packets from queue in host
- deferral can happen to ksoftirqd
- Cilium's eBPF prog called only on tc ingress to redirect to phys dev

Cilium: meta devices for eBPF

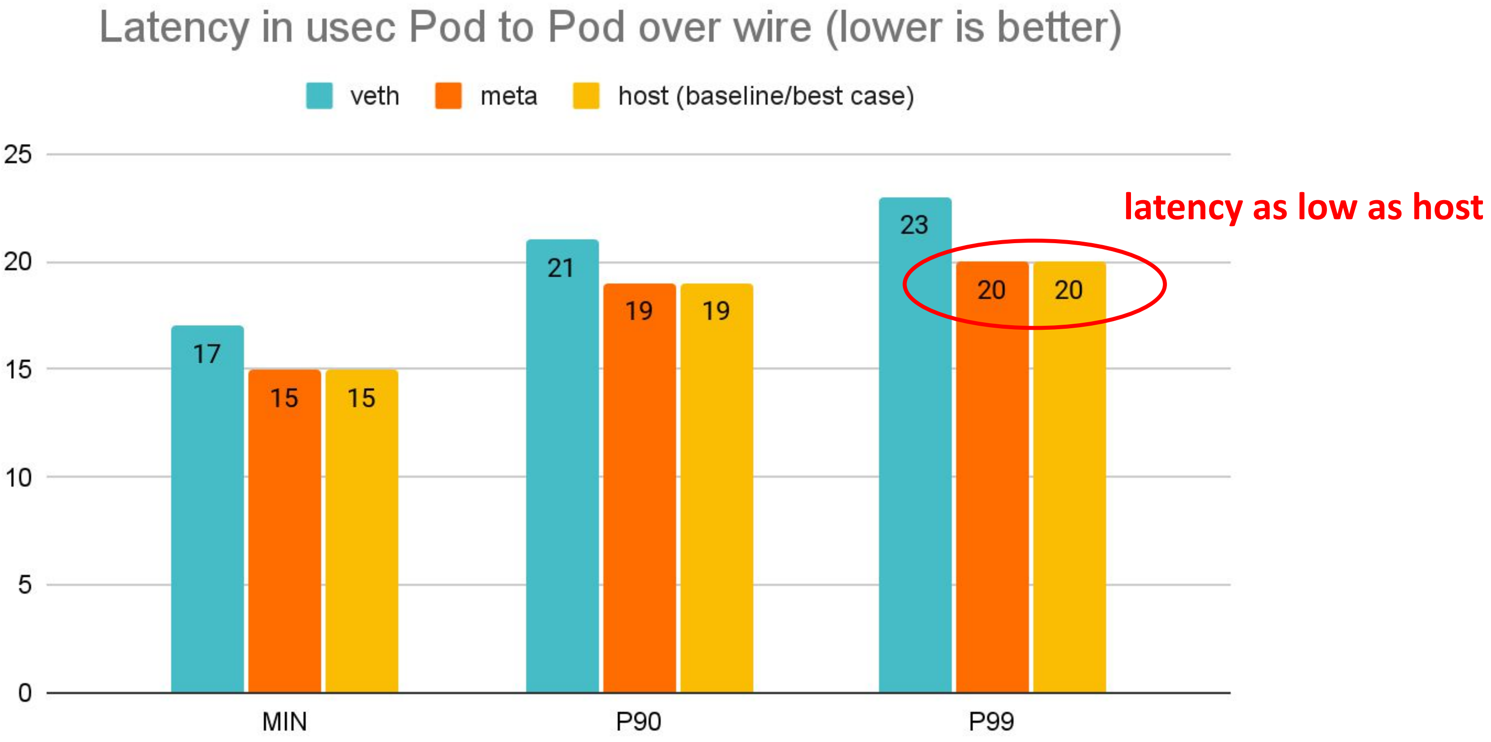


Internals for meta (new):

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

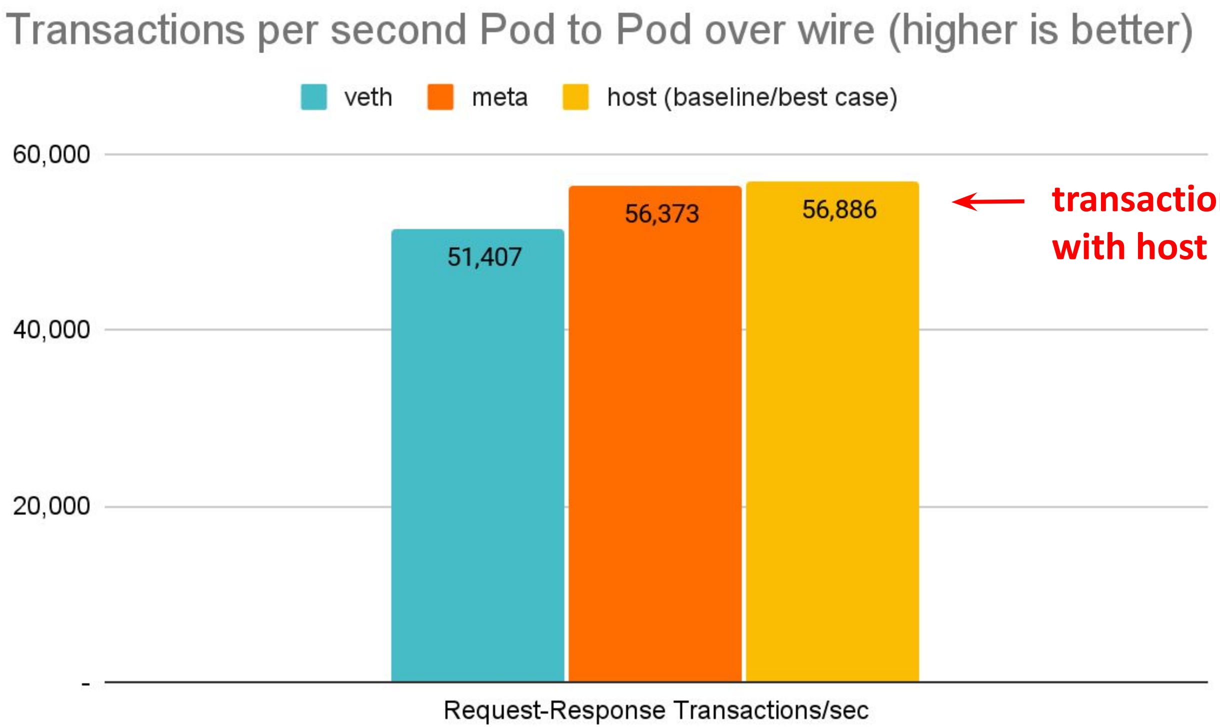


Cilium: meta devices for eBPF



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

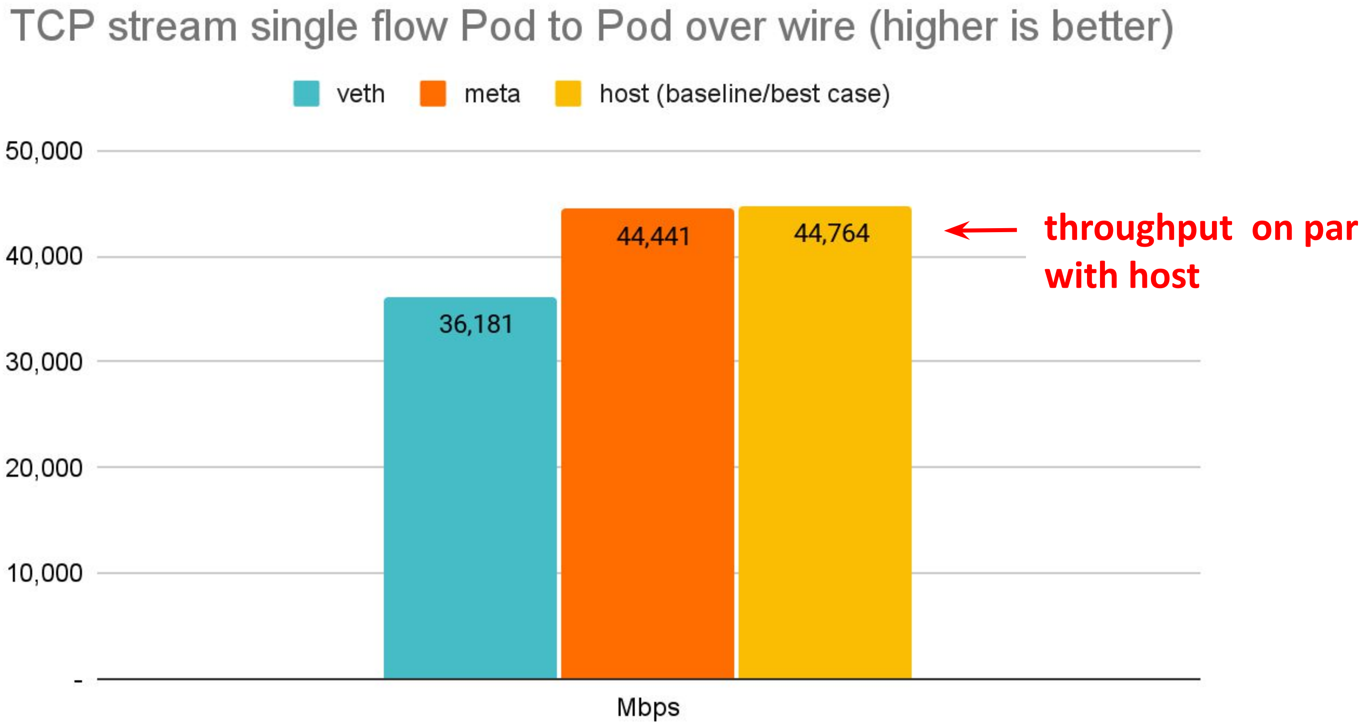
Cilium: meta devices for eBPF



transaction rate on par with host

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

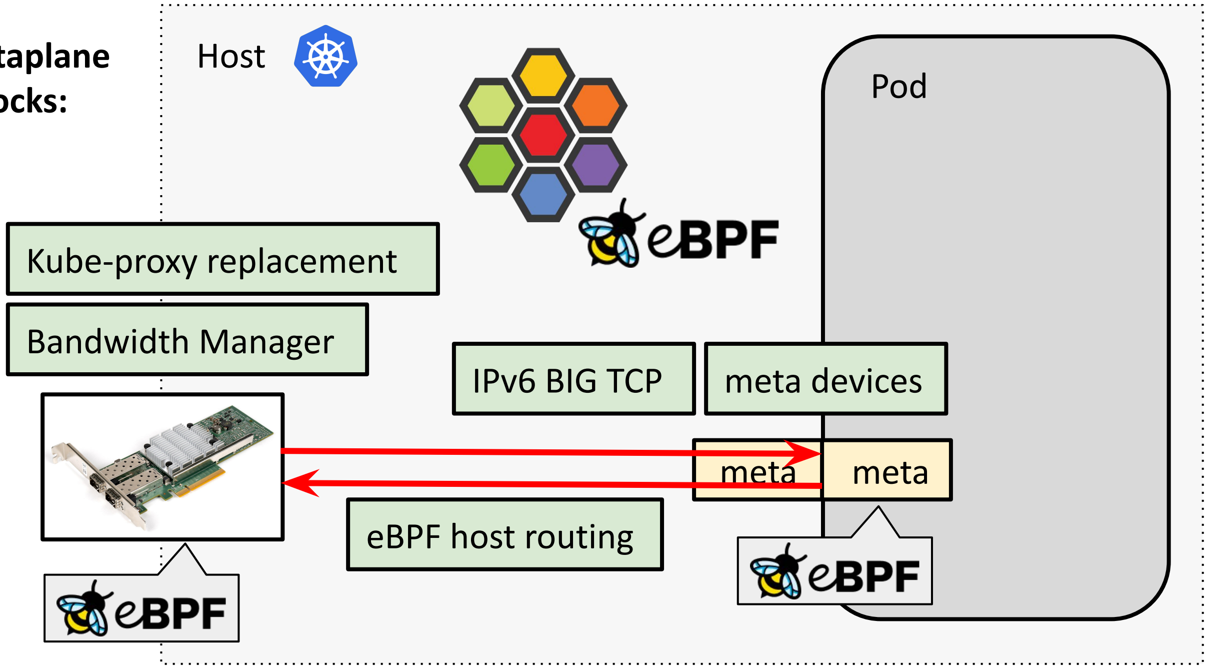
Cilium: meta devices for eBPF



Back to back: AMD Ryzen 9 3950X @ 3.5 GHz, 128G RAM @ 3.2 GHz, PCIe 4.0, ConnectX-6 Dx, mlx5 driver
netperf -t TCP_STREAM -H <remote pod> -l 60

Cilium: Tomorrow's Data Plane - Summary

Cilium's dataplane building blocks:



Acknowledgements

- ➔ Eric Dumazet
- ➔ Coco Li
- ➔ Yuchung Cheng
- ➔ Martin Lau
- ➔ John Fastabend
- ➔ K8s, Cilium, BPF & netdev kernel community

ISOVALENT

Thank you! Questions?

github.com/cilium/cilium

cilium.io

ebpf.io

Isovalent: booth S35

Cilium: kiosk 21 (@ project pavilion)

tl;dr: Covered Cilium feature preview:

- IPv6 ‘BIG TCP’ support for Pods
- Stateful NAT46/64 Gateway
- Stateless NAT46/64 Gateway
- meta network driver for Pods as low-latency veth replacement

P.S.: While preparing the talk, turns out there is progress.