
KubeCon
CloudNativeCon
OPEN SOURCE SUMMIT
China 2023


KubeCon | CloudNativeCon

OPEN SOURCE SUMMIT

China 2023

Unleashing the Magic: Harnessing eBPF for Traffic Redirection in Istio Ambient Mode

Chun Li & Iris Ding, Intel

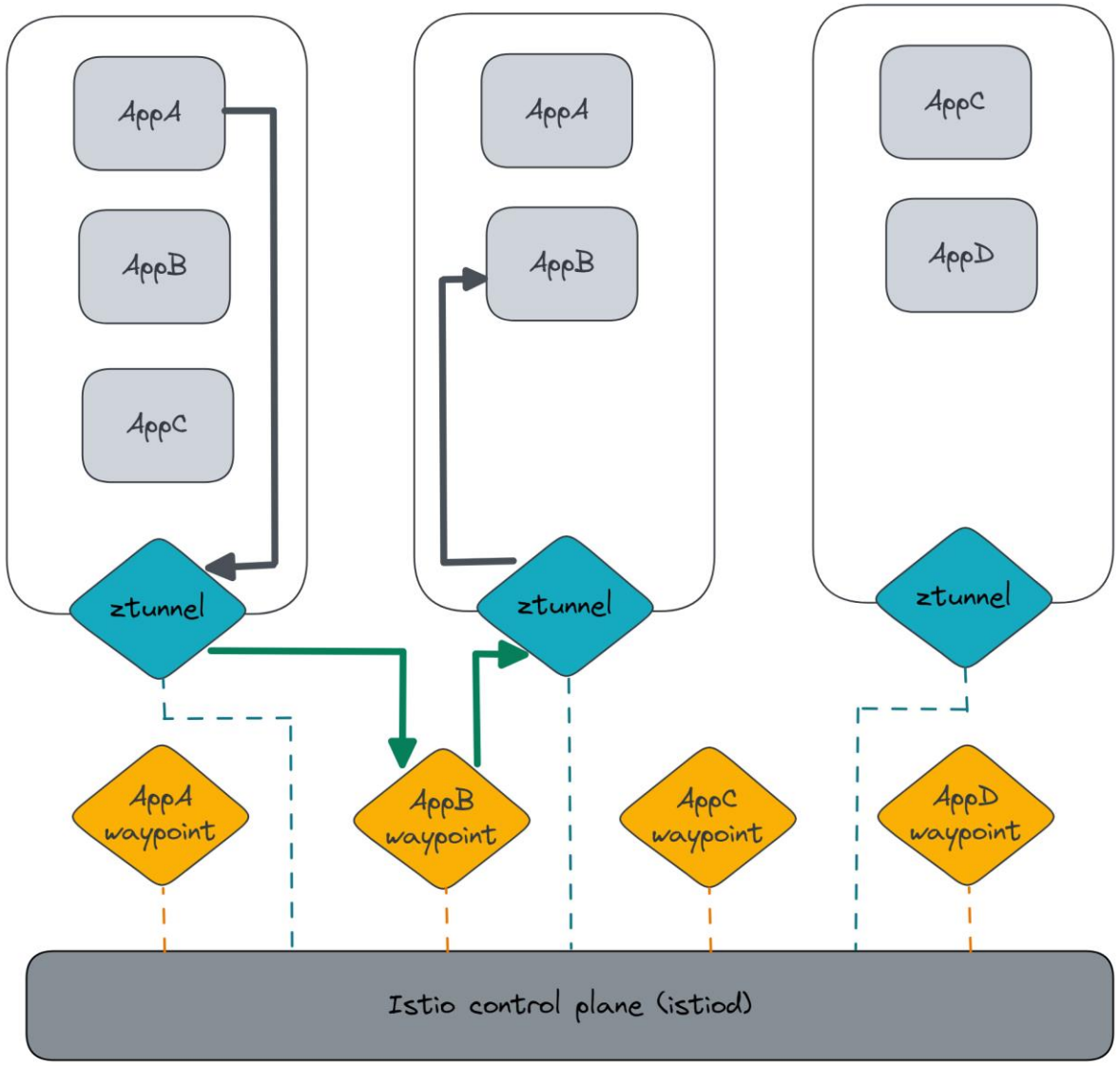
Outline

- What is ambient mesh
- How ambient mesh works
- How eBPF reinforces ambient mesh
- Next(lesson learned)…

Background

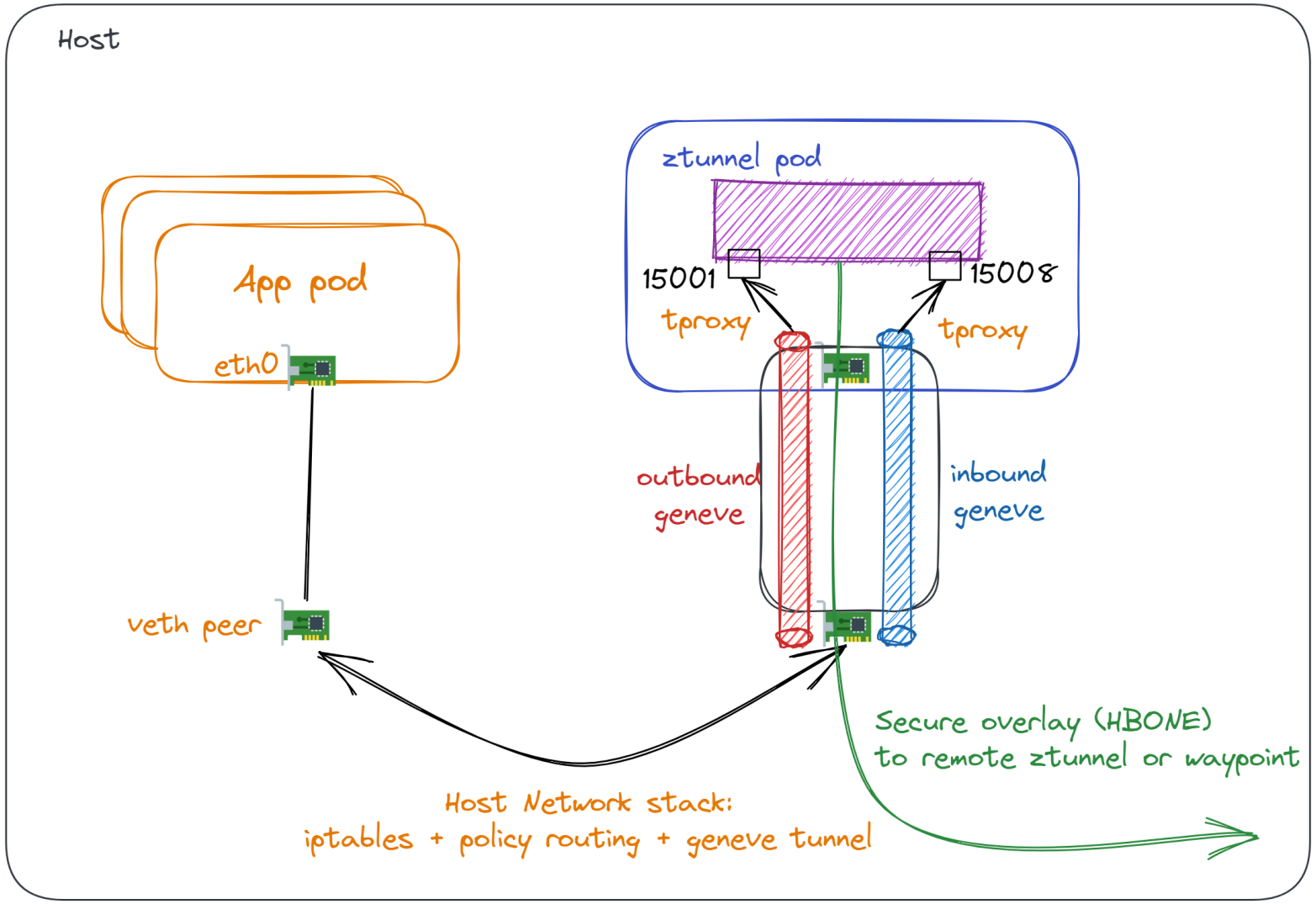
- Ambient mesh is a new Istio data plane mode
- Launched in Sep. 2022. announcement
- Currently in alpha status, moving forward to beta phase

What is ambient mesh



- Removing the data plane from the application pod
- Istio ambient does not use sidecars
- Separates out L4 capabilities from L7
- Scales L7 proxies on demand

How ambient mesh works



How & Why?

Iptables

+

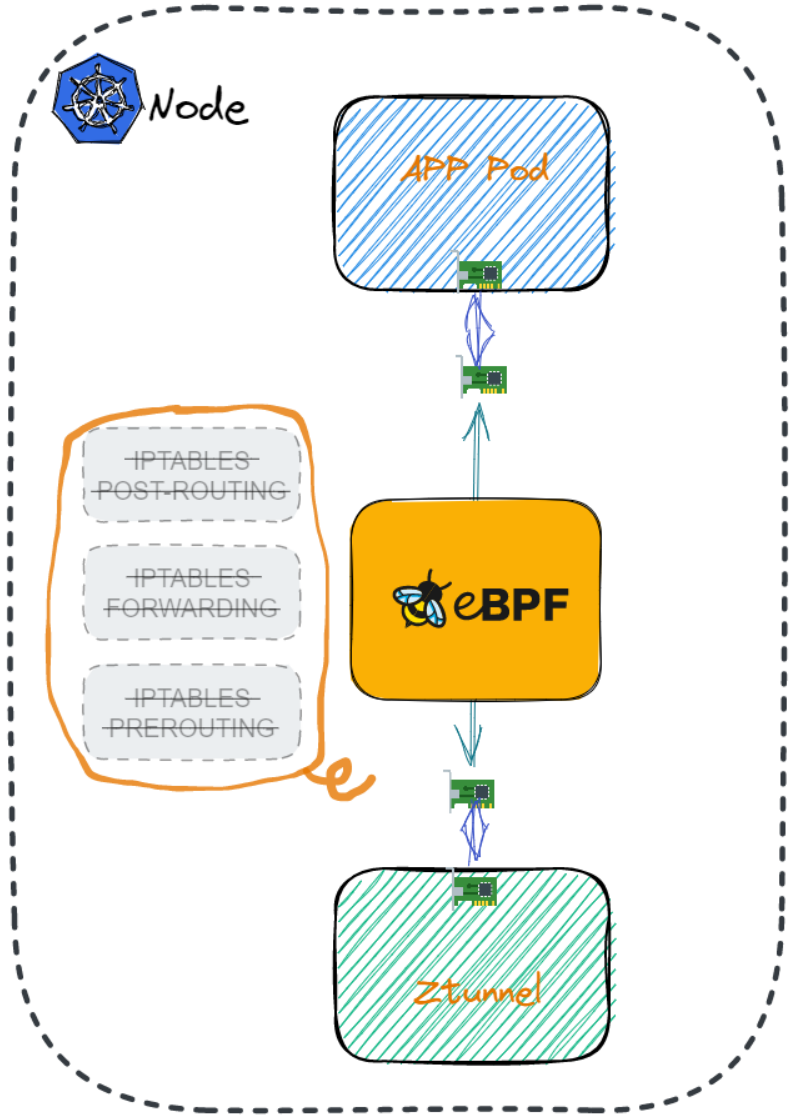
Policy Routing

+

Overlay tunnel

Intercepts/Redirects traffic transparently

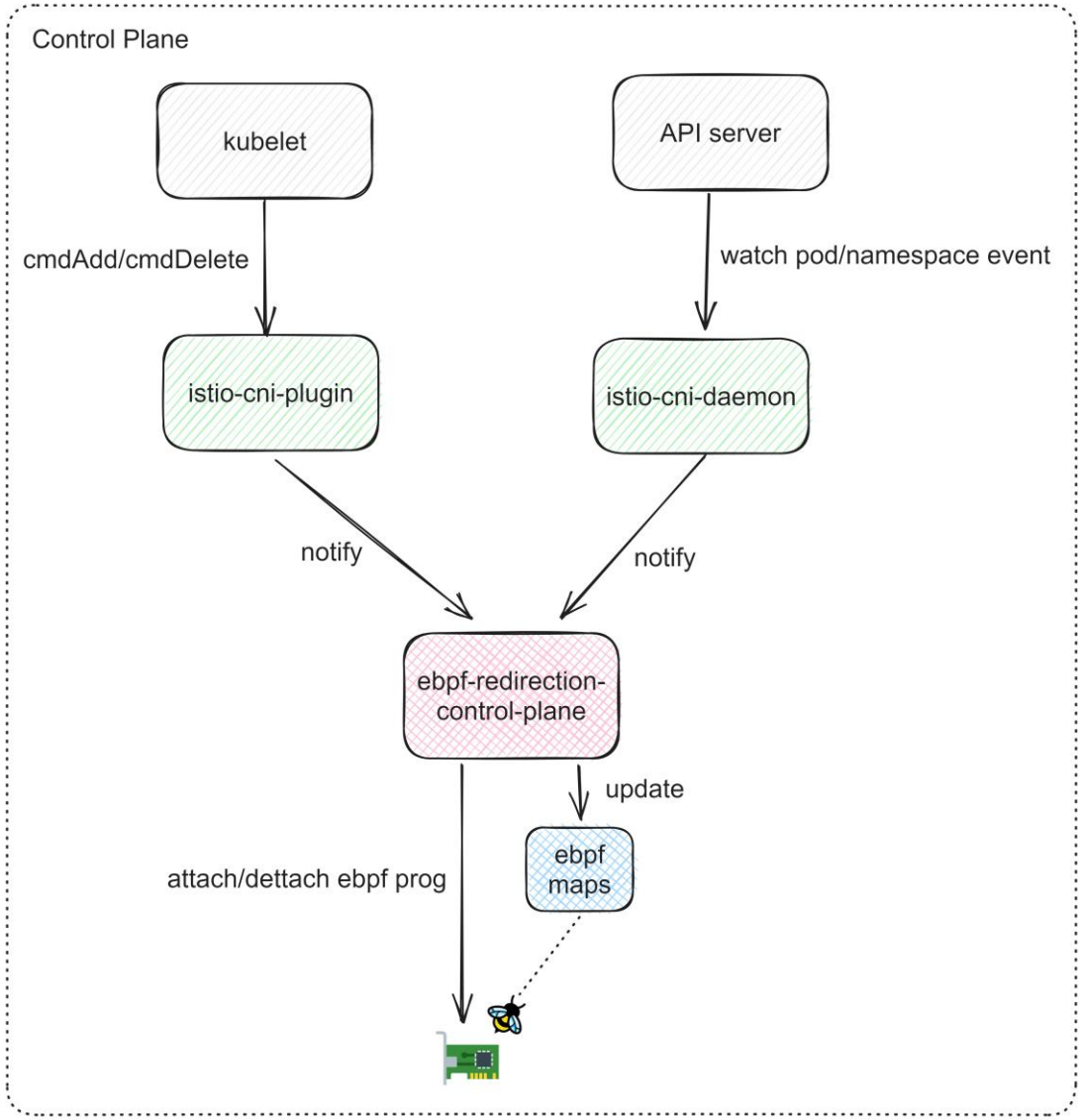
How eBPF reinforces ambient mesh



eBPF simplifies the redirection

With eBPF, if a packet shows up at point A, and you know that the packet needs to go to point B, you can optimize the network path in the Linux kernel, so that packets bypass complex routing and simply arrive at their final destination.

How eBPF reinforces ambient mesh



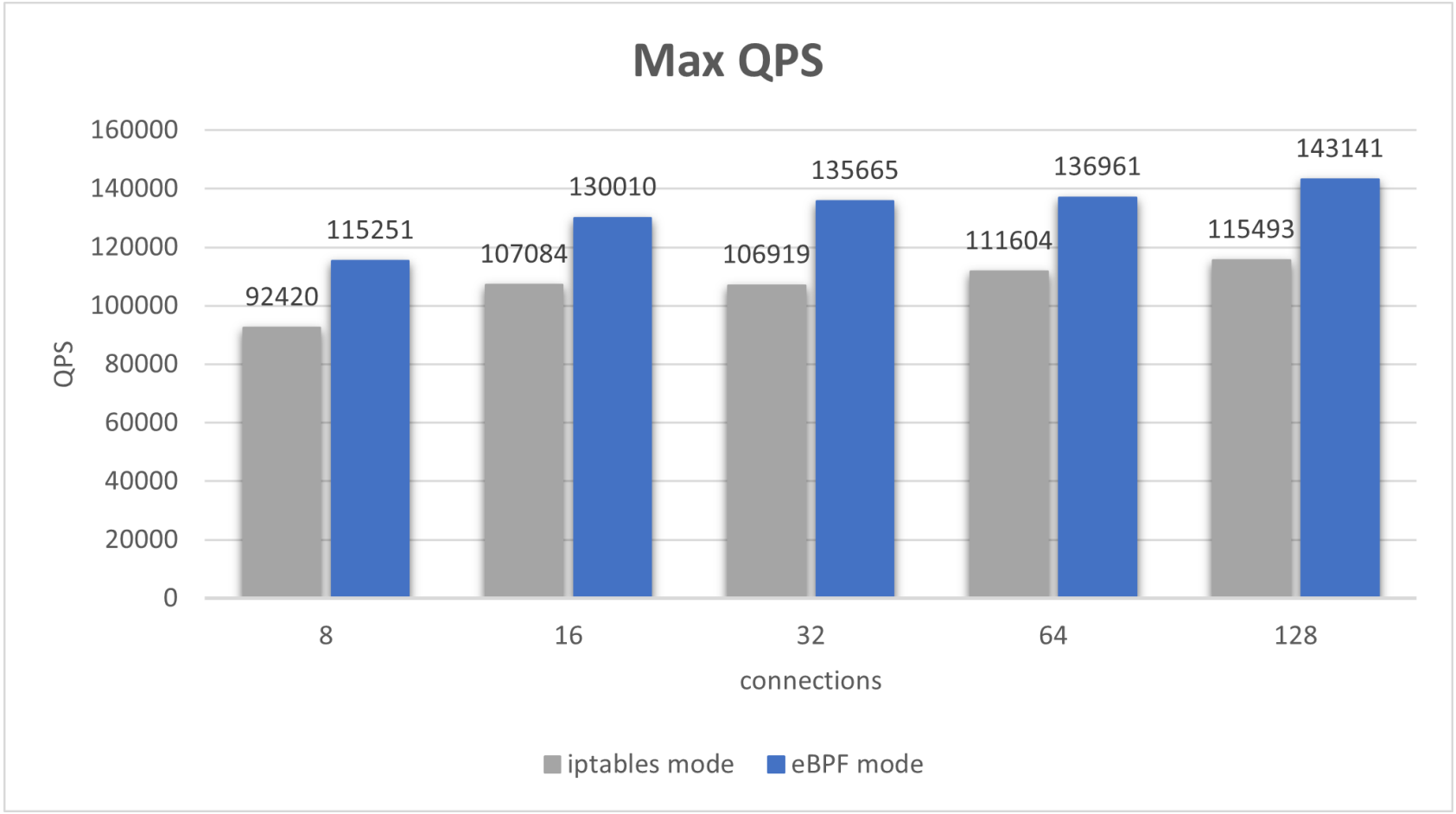
Control plane

istio-cni-plugin will notify redirection-control-plane once a pod's interface is added/deleted, redirection-control-plane will be responsible for attaching/detaching an eBPF program to the corresponding interface.

istio-cni will continuously watch namespace and pod events on its node. Similar to cni-plugin case, attachment/detachment will be performed dynamically by redirection-control-plane.

eBPF-map acts as "bridge" between userspace(control plane) and kernel space(dataplane)

How eBPF reinforces ambient mesh



Tests run in a kind cluster with a Fortio client sending requests to a Fortio server on the same Kubernetes worker node.

How to enable

1. Follow the instructions in [Getting Started with Ambient Mesh]

2. In istio install stage set the mode to ebpf by ' *--set values.cni.ambient.redirectMode="ebpf"* ', e.g.:

   istioctl install --set profile=ambient --skip-confirmation --set values.cni.ambient.redirectMode="ebpf"

3. Check the istio-cni logs to confirm eBPF redirection is on:

   ambient Writing ambient config: {"ztunnelReady":true,"redirectMode":"ebpf"}

```
root@kind-worker2:/# tc filter show dev vetha8baedab ingress
filter protocol all pref 1 bpf chain 0
filter protocol all pref 1 bpf chain 0 handle 0x1 ztunnel_host_in direct-action not_in_hw id 342 tag c47256f698027770 jited
root@kind-worker2:/# tc filter show dev vethfb85c476 ingress
filter protocol all pref 1 bpf chain 0
filter protocol all pref 1 bpf chain 0 handle 0x1 app_outbound direct-action not_in_hw id 341 tag 8ddfa96e677d47f3 jited
root@kind-worker2:/# tc filter show dev vethfb85c476 egress
filter protocol all pref 1 bpf chain 0
filter protocol all pref 1 bpf chain 0 handle 0x1 app_inbound direct-action not_in_hw id 340 tag 9350368009a53570 jited
```

Current status

Unfortunately, eBPF support is temporarily disabled and pending CNCF establishing guidance around dual-licensed eBPF bytecode cncf/toc#1000 (comment)

However, you could build the image with git repo

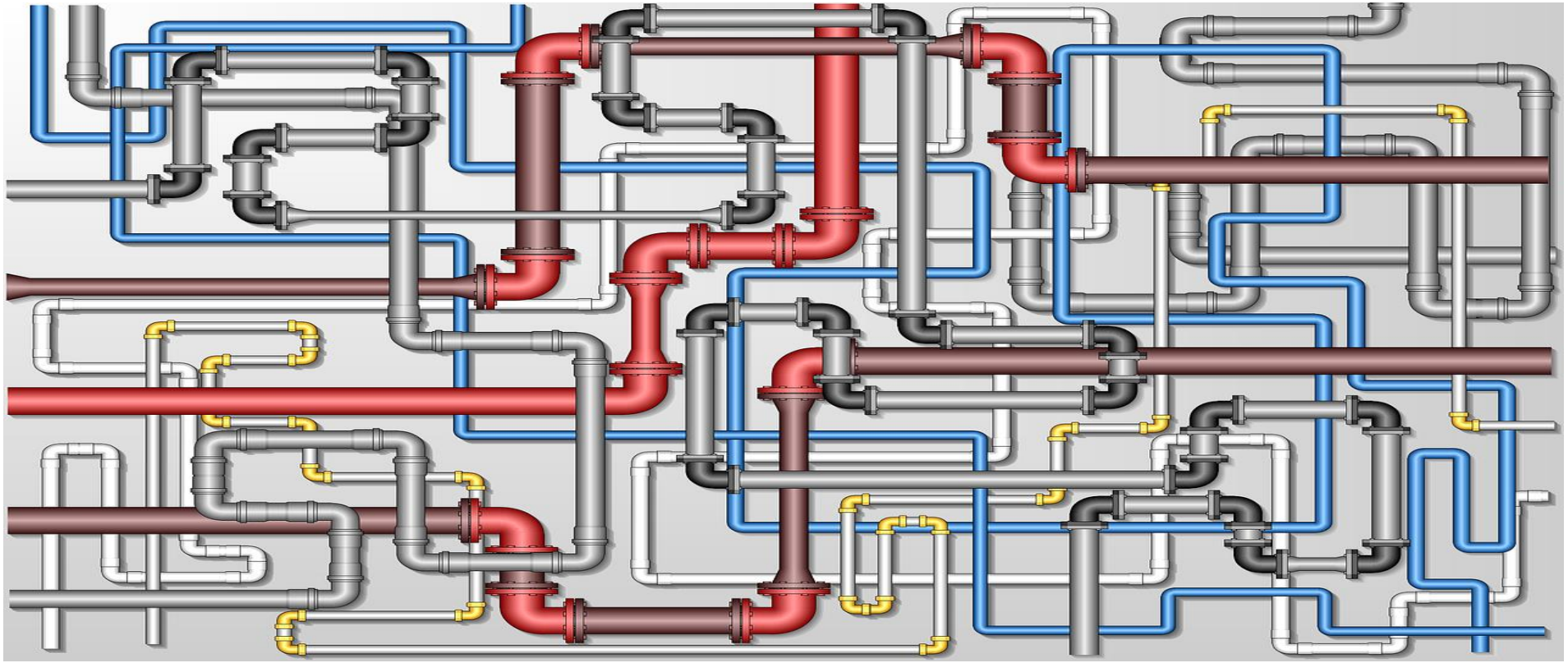
git clone https://github.com/istio/istio.git; cd istio
git checkout 1.19.0
git revert 94c639a2dad33a1168aa5f399ad780643c715205
tools/docker --no-cache --targets=install-cni --hub=127.0.0.1:5000 --tag=1.19.0-ebpf

Or directly use a prebuild docker image platform934/install-cni:1.19.0-ebpf (for experiment only!)

istioctl install --set profile=ambient --skip-confirmation --set values.cni.ambient.redirectMode="ebpf" --set values.cni.hub=docker.io/platform934 --set values.cni.tag=1.19.0-ebpf

Next...

How will things go on while multiple components are emerging with tc eBPF?

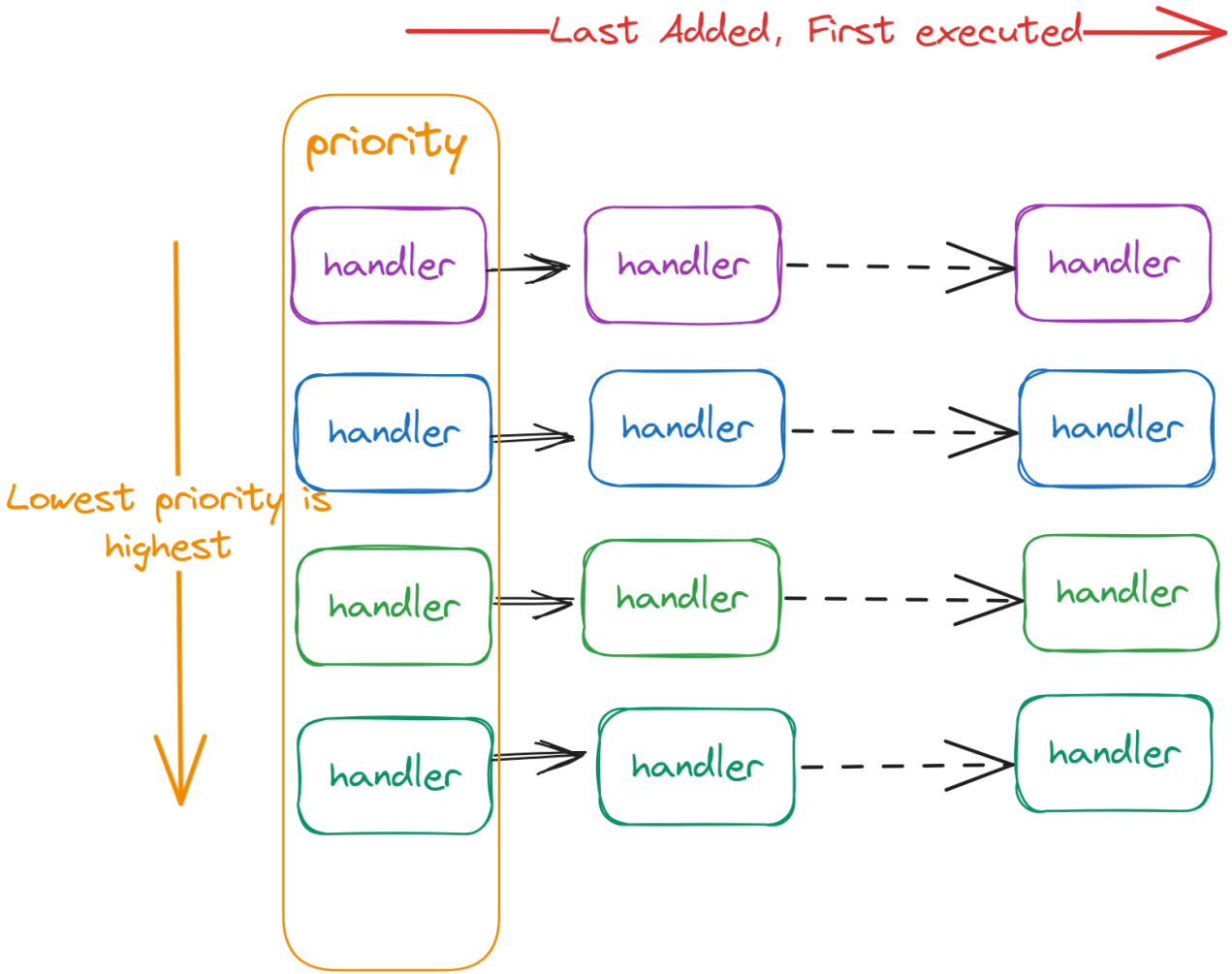


Common example snippet for tc eBPF


```c
SEC("tc")
int tc_ingress(struct __sk_buff *ctx)
{
        void *data_end = (void *)(__u64)ctx->data_end;
        void *data = (void *)(__u64)ctx->data;
        struct ethhdr *l2;
        struct iphdr *l3;

        if (ctx->protocol != bpf_htons(ETH_P_IP))
                return TC_ACT_OK;

        l2 = data;
        if ((void *)(l2 + 1) > data_end)
                return TC_ACT_OK;

        l3 = (struct iphdr *)(l2 + 1);
        if ((void *)(l3 + 1) > data_end)
                return TC_ACT_OK;

        bpf_printk("Got IP packet: tot_len: %d, ttl: %d", bpf_ntohs(l3->tot_len), l3->ttl);
        return TC_ACT_OK;
}
```

```c
SEC("drop_non_tun_vip")
int _drop_non_tun_vip(struct __sk_buff *skb)
{
        struct bpf_tunnel_key tkey = {};
        void *data = (void *)(long)skb->data;
        struct eth_hdr *eth = data;
        void *data_end = (void *)(long)skb->data_end;

        if (data + sizeof(*eth) > data_end)
                return TC_ACT_OK;

        if (eth->h_proto == htons(ETH_P_IP)) {
                struct iphdr *iph = data + sizeof(*eth);

                if (data + sizeof(*eth) + sizeof(*iph) > data_end)
                        return TC_ACT_OK;

                if (is_vip_addr(eth->h_proto, iph->daddr))
                        return TC_ACT_SHOT;
        } else if (eth->h_proto == htons(ETH_P_IPV6)) {
                struct ipv6hdr *ip6h = data + sizeof(*eth);

                if (data + sizeof(*eth) + sizeof(*ip6h) > data_end)
                        return TC_ACT_OK;

                if (is_vip_addr(eth->h_proto, ip6h->daddr.s6_addr32[0]))
                        return TC_ACT_SHOT;
        }

        return TC_ACT_OK;
}
```

Traffic Control



For each handler, the return codes are defined in linux/pkt_cls.h

TC_ACT_OK (0) , will terminate the packet processing pipeline and allows the packet to proceed

TC_ACT_SHOT (2) , will terminate the packet processing pipeline and drops the packet

TC_ACT_UNSPEC (-1) , will use the default action configured from tc (similarly as returning -1 from a classifier)

TC_ACT_PIPE (3) , will iterate to the next action, if available

Traffic Control


```c
__section("tc1ok")
int hook1_ok(struct __sk_buff *skb)
{
    dbg("hook1_ok handler processed\n");
    return TC_ACT_OK;
}

__section("tc2ok")
int hook2_ok(struct __sk_buff *skb)
{
    dbg("hook2_ok handler processed\n");
    return TC_ACT_OK;
}

__section("tc1pipe")
int hook1_pipe(struct __sk_buff *skb)
{
    dbg("hook1_pipe handler processed\n");
    return TC_ACT_PIPE; // same effect as TC_ACT_UNSPEC
}

__section("tc2pipe")
int hook2_pipe(struct __sk_buff *skb)
{
    dbg("hook2_pipe handler processed\n");
    return TC_ACT_PIPE; // same effect as TC_ACT_UNSPEC
}
```

Hook returns ***TC_ACT_OK***

```
filter protocol all pref 1 bpf chain 0
filter protocol all pref 1 bpf chain 0 handle 0x1 tc_ebpf_example.o:[tc1ok] direct-action not
filter protocol all pref 2 bpf chain 0
filter protocol all pref 2 bpf chain 0 handle 0x1 tc_ebpf_example.o:[tc2ok] direct-action not
```

Only **hook1_ok** is called

Debug output:

```
node-1485099 [000] ..s1 1709204.924715: 0: hook1_ok handler processed
sshd-1485302 [004] ..s1 1709204.934063: 0: hook1_ok handler processed
node-1485356 [006] ..s1 1709204.935635: 0: hook1_ok handler processed
node-1485356 [006] ..s1 1709204.935643: 0: hook1_ok handler processed
node-1485099 [002] ..s1 1709205.385850: 0: hook1_ok handler processed
node-1485099 [002] ..s1 1709205.385861: 0: hook1_ok handler processed
sshd-1485302 [003] ..s1 1709205.406107: 0: hook1_ok handler processed
node-1485356 [000] ..s1 1709205.407033: 0: hook1_ok handler processed
node-1485356 [000] ..s1 1709205.407051: 0: hook1_ok handler processed
node-1485099 [002] ..s1 1709205.485918: 0: hook1_ok handler processed
node-1485099 [002] ..s1 1709205.485931: 0: hook1_ok handler processed
node-1485356 [000] ..s1 1709205.490380: 0: hook1_ok handler processed
node-1485356 [000] ..s1 1709205.490390: 0: hook1_ok handler processed
sshd-1485302 [003] ..s1 1709205.507348: 0: hook1_ok handler processed
node-1485356 [000] ..s1 1709205.508983: 0: hook1_ok handler processed
sshd-1485302 [003] ..s1 1709205.509000: 0: hook1_ok handler processed
```

Traffic Control


```c
__section("tc1ok")
int hook1_ok(struct __sk_buff *skb)
{
    dbg("hook1_ok handler processed\n");
    return TC_ACT_OK;
}

__section("tc2ok")
int hook2_ok(struct __sk_buff *skb)
{
    dbg("hook2_ok handler processed\n");
    return TC_ACT_OK;
}

__section("tc1pipe")
int hook1_pipe(struct __sk_buff *skb)
{
    dbg("hook1_pipe handler processed\n");
    return TC_ACT_PIPE; // same effect as TC_ACT_UNSPEC
}

__section("tc2pipe")
int hook2_pipe(struct __sk_buff *skb)
{
    dbg("hook2_pipe handler processed\n");
    return TC_ACT_PIPE; // same effect as TC_ACT_UNSPEC
}
```

Hook returns ***TC_ACT_PIPE/TC_ACT_UNSPEC***

```
filter protocol all pref 1 bpf chain 0
filter protocol all pref 1 bpf chain 0 handle 0x2 tc_ebpf_example.o:[tc2pipe] direct-action
filter protocol all pref 1 bpf chain 0 handle 0x1 tc_ebpf_example.o:[tc1pipe] direct-action
filter protocol all pref 2 bpf chain 0
filter protocol all pref 2 bpf chain 0 handle 0x2 tc_ebpf_example.o:[tc1pipe] direct-action
filter protocol all pref 2 bpf chain 0 handle 0x1 tc_ebpf_example.o:[tc2pipe] direct-action
```

Debug output:

```
          node-1485099 [006] ..s1 1709857.627827: 0: hook2_pipe handler processed
          sshd-1485302 [007] ..s1 1709857.629965: 0: hook2_pipe handler processed
          sshd-1485302 [007] ..s1 1709857.629969: 0: hook1_pipe handler processed
          sshd-1485302 [007] ..s1 1709857.629969: 0: hook1_pipe handler processed
          sshd-1485302 [007] ..s1 1709857.629970: 0: hook2_pipe handler processed
    Xtigervnc-352401   [007] ..s. 1709857.670196: 0: hook2_pipe handler processed
    Xtigervnc-352401   [007] ..s. 1709857.670201: 0: hook1_pipe handler processed
    Xtigervnc-352401   [007] ..s. 1709857.670201: 0: hook1_pipe handler processed
    Xtigervnc-352401   [007] ..s. 1709857.670202: 0: hook2_pipe handler processed
          sshd-1485302 [007] ..s1 1709857.769226: 0: hook2_pipe handler processed
          sshd-1485302 [007] ..s1 1709857.769235: 0: hook1_pipe handler processed
          sshd-1485302 [007] ..s1 1709857.769236: 0: hook1_pipe handler processed
          sshd-1485302 [007] ..s1 1709857.769237: 0: hook2_pipe handler processed
```

Next...

There is still plenty of work to be done:

- Integration with various CNI plugins
- Interact with Network Policies
- Acceleration with sockmap
- …

Contributions to improve the ease of use would be greatly welcomed.

Join us in #ambient on the Istio slack.


KubeCon
CloudNativeCon
OPEN SOURCE SUMMIT
China 2023
Thank you!