All Your Queues Are Belong to Us

Debugging and Mitigating a Kernel Bug with eBPF

Laurent Bernaille

DATADOG

About Datadog

Over 500 integrations
Over 3000 employees
Over 18,500 customers
Millions of hosts reporting
Trillions of data points per day

10000s hosts in our infra
Dozens of k8s clusters
Multi-cloud
Rapid growth

DATADOG

eBPF @Datadog

Product: Observability

Security events
Network performance
Universal Service Discovery
Growing fast

Infrastructure

Container networking: Cilium
Debugging: bpftrace

DATADOG

It all started with a performance regression

Networking issues in a few clusters

- Two applications impacted (at least)
- Symptoms
  - Packet drops and retransmits
  - Lower throughput
  - Increased latency
- Consequences: more processing pods, more expensive
- Only some clusters impacted: old clusters seem fine

After some time, we found this

```
$ ethtool -S ens6 | grep tx_bytes
     queue_0_tx_bytes: 221945535332O3
     queue_1_tx_bytes: 0
     queue_2_tx_bytes: 0
     queue_3_tx_bytes: 0
     queue_4_tx_bytes: 0
     queue_5_tx_bytes: 0
     queue_6_tx_bytes: 0
     queue_7_tx_bytes: 0
```

We have 8 tx queues on the Network interface of this instance
We only use the first one

DATADOG

Diving a bit deeper

```
$ sudo tc -s qdisc show dev ens6
qdisc mq 0: root
 Sent 21809447867 bytes 15853650 pkt (dropped 279030, overlimits 0 requeues 521)
 backlog 0b 0p requeues 212

qdisc fq_codel 0: parent :1 limit 10240p flows 1024 [...]
 Sent 21809447867 bytes 15853650 pkt (dropped 279030, overlimits 0 requeues 521)
 [...]

qdisc fq_codel 0: parent :2 limit 10240p flows 1024 [...]
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 100)
 [...]

[+ 1 stanza for each remaining queue]
```

- mq qdisc: multiqueue, 1 qdisc per tx queue
- only first qdisc queue is used
- fq_codel => limited throughput triggers queuing, which triggers drops (~2% here!)
- Mitigation: replace fq_codel with fq (not a real fix)

DATADOG

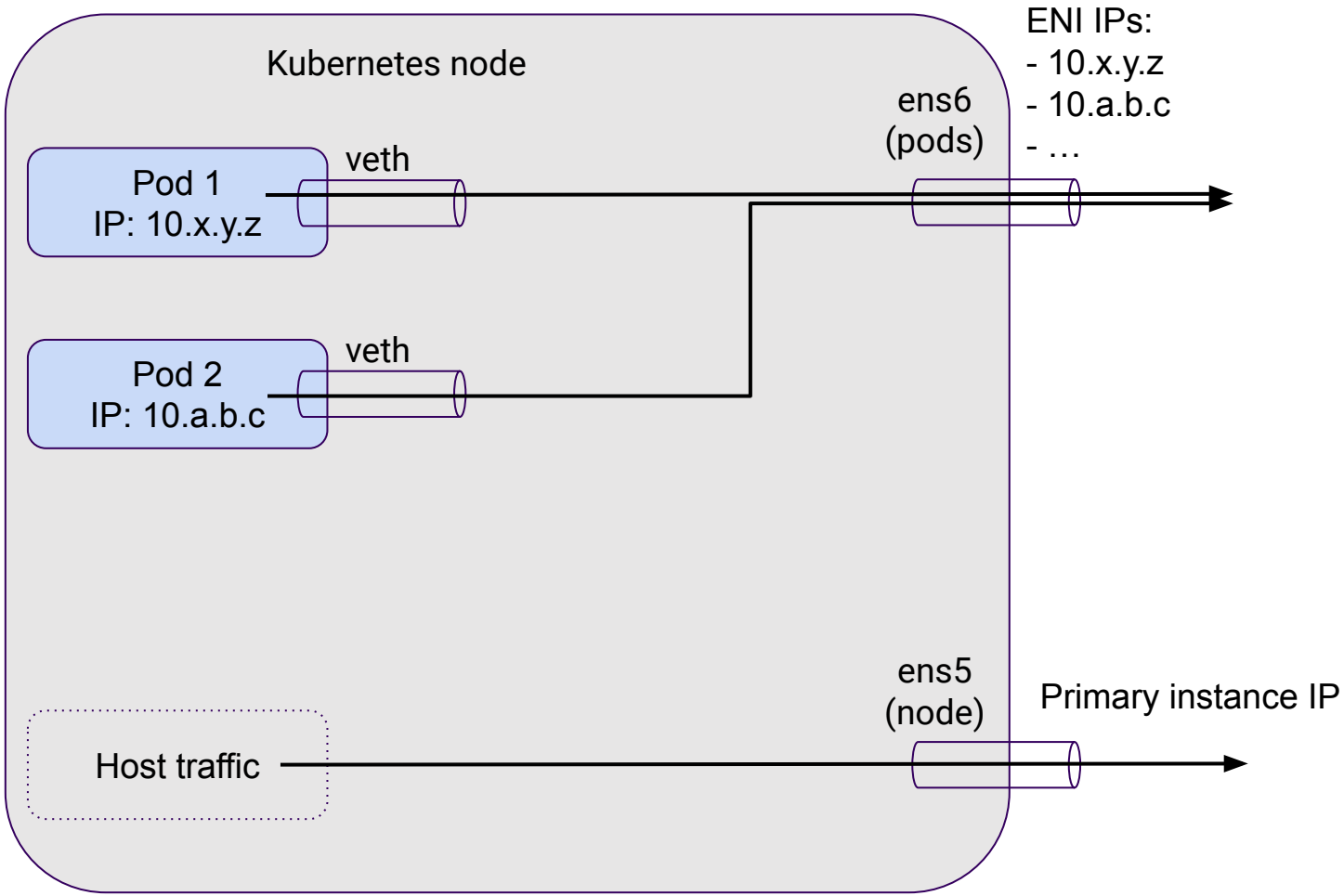
What about rx?

```
$ ethtool -S ens6 | grep rx_bytes
     queue_0_rx_bytes: 25881756
     queue_1_rx_bytes: 6018503
     queue_2_rx_bytes: 15896451
     queue_3_rx_bytes: 2289972
     queue_4_rx_bytes: 5496205
     queue_5_rx_bytes: 14631267
     queue_6_rx_bytes: 9710508
     queue_7_rx_bytes: 1385099
```

rx is OK: we use all the queues

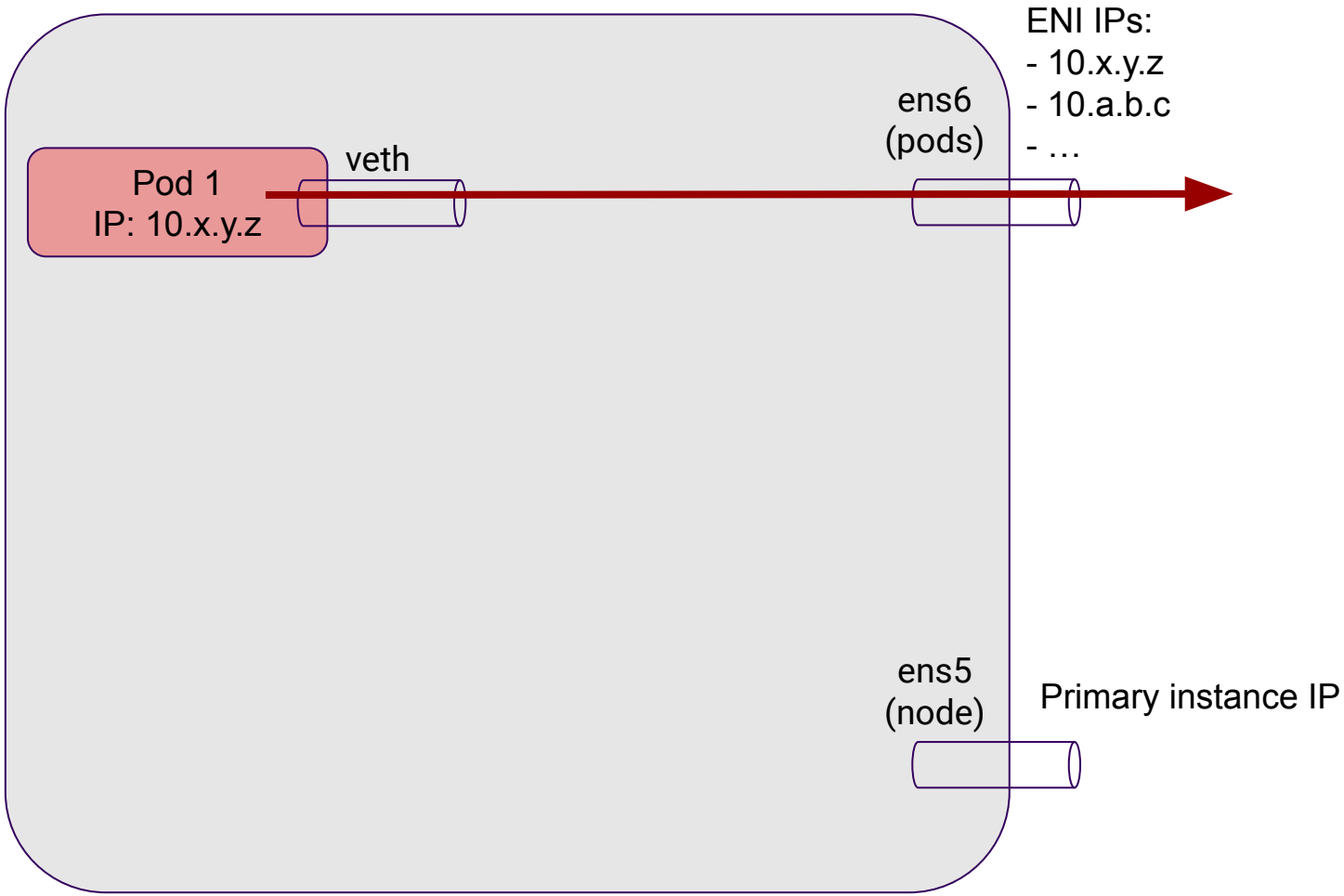
DATADOG

Our networking setup: Cilium IPAM=eni mode

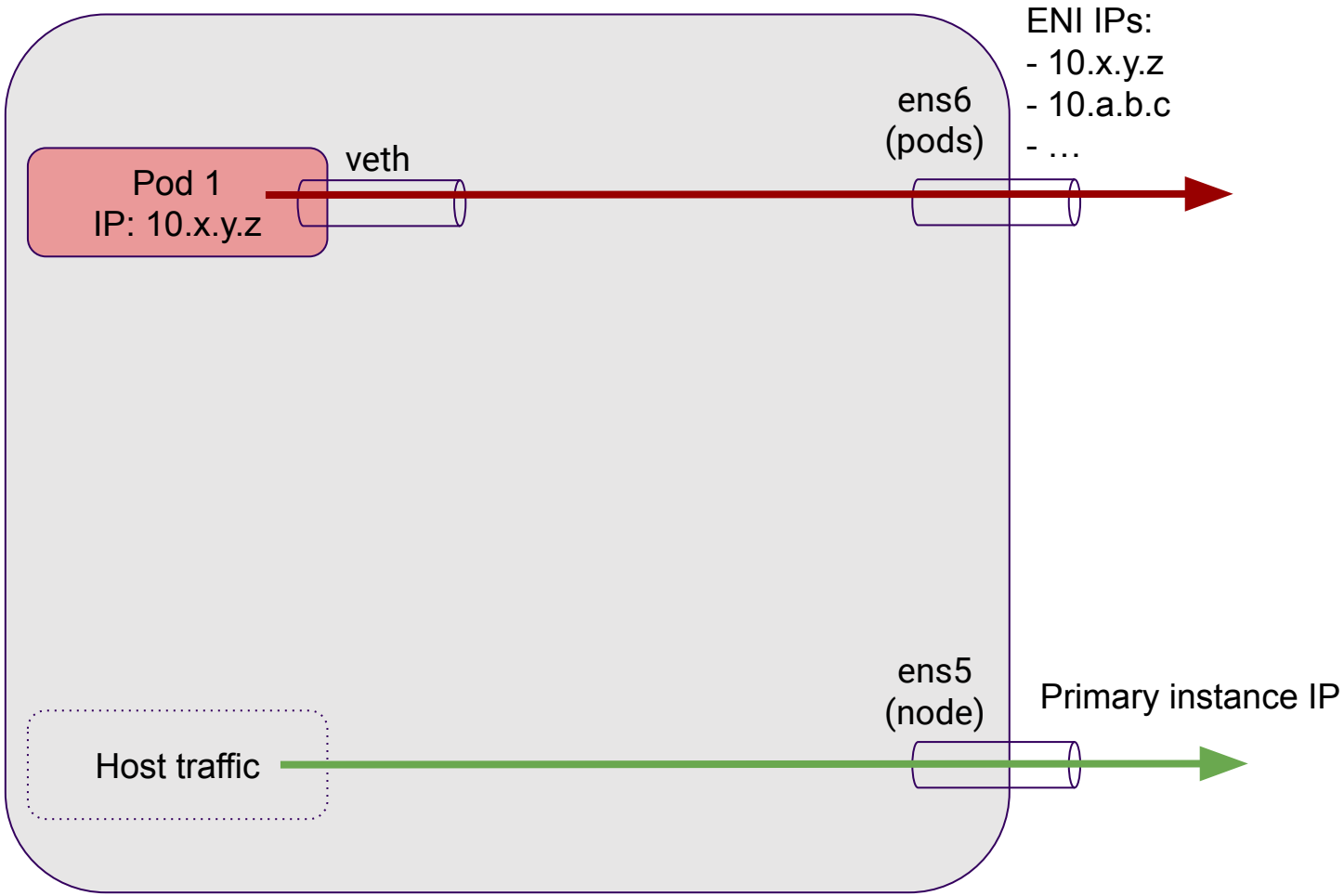


DATADOG

Our issue impacts pod egress traffic

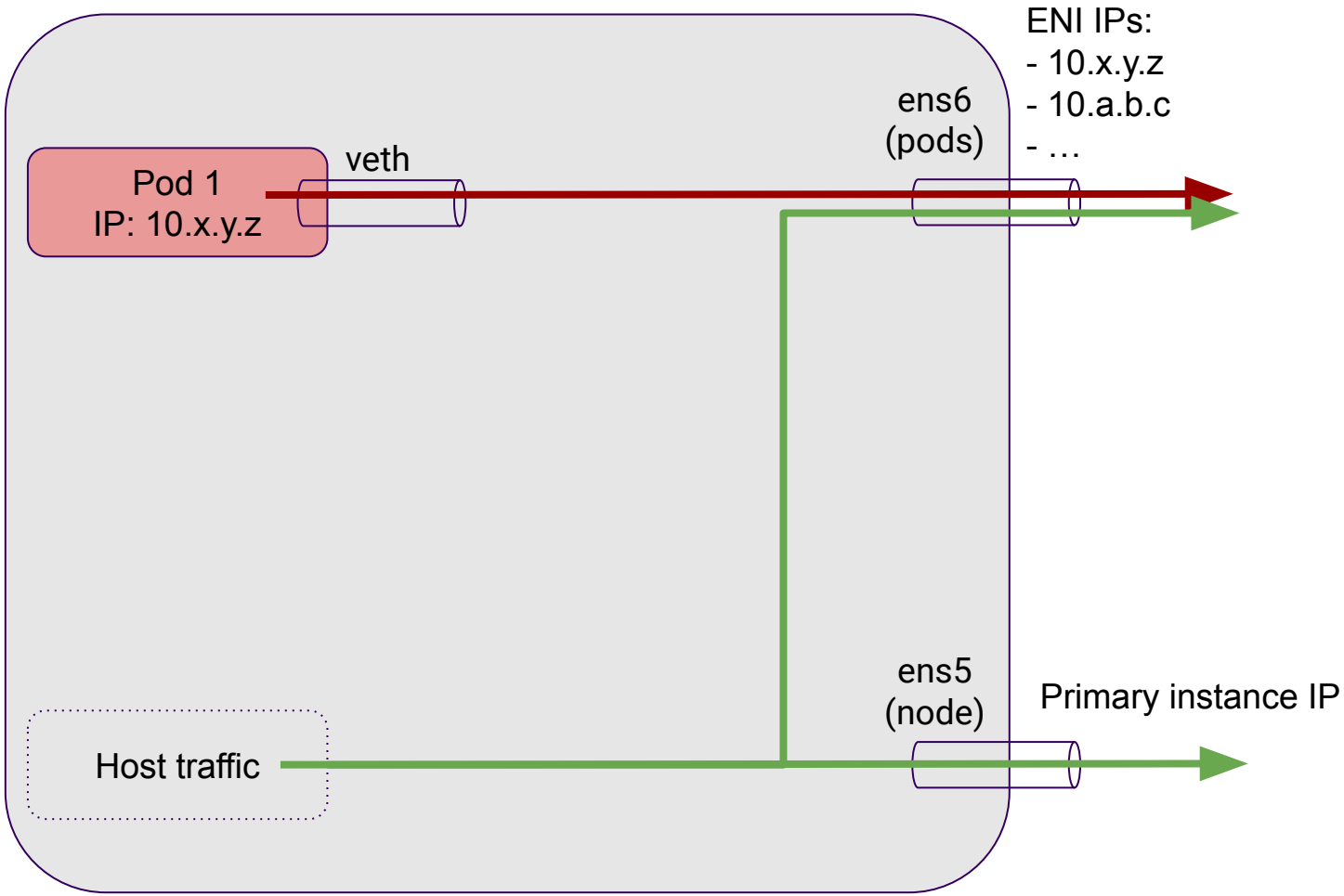


What about host traffic?



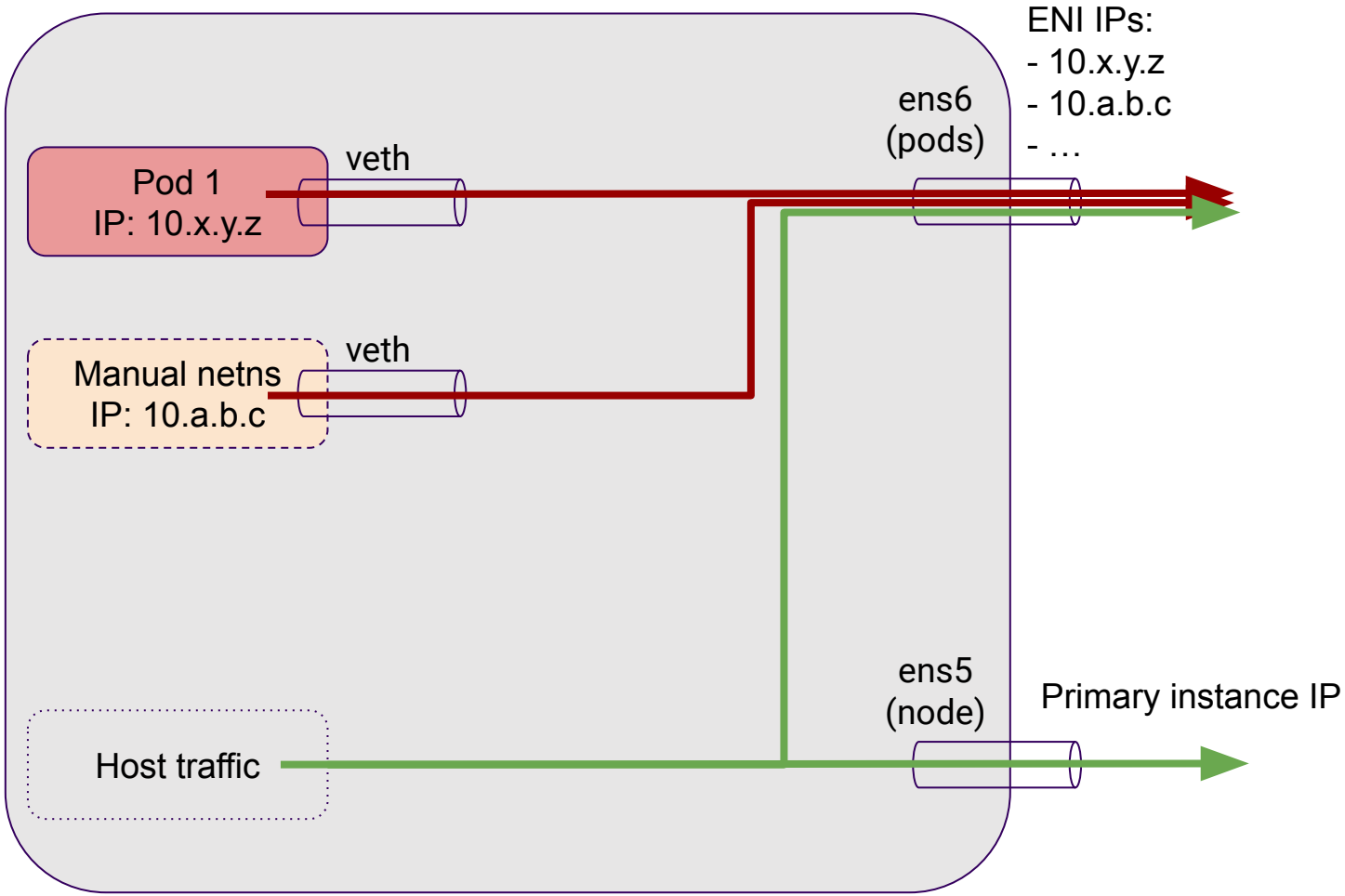
DATADOG

What about host traffic going through ens6?



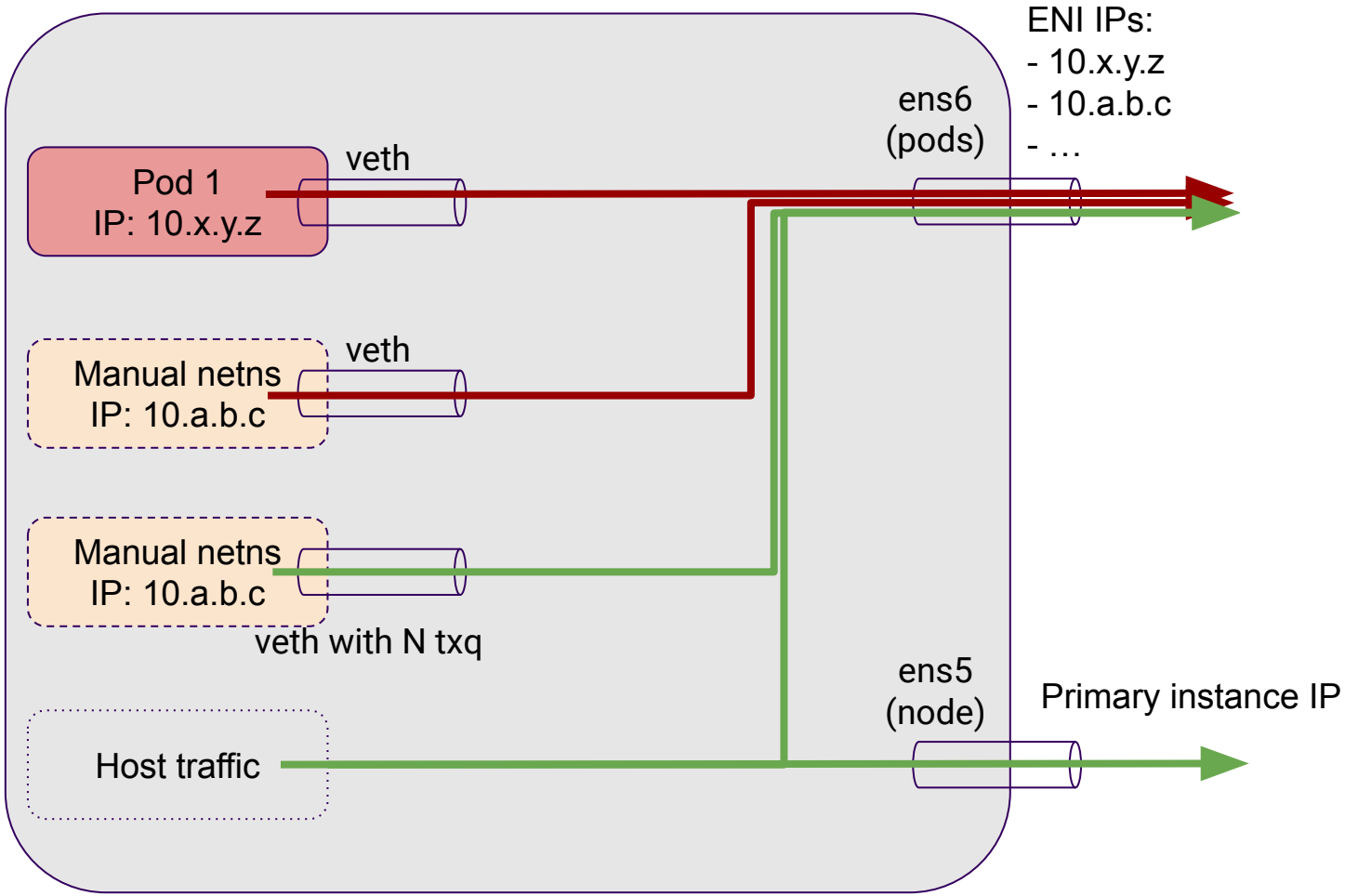
DATADOG

Could it be our CNI plugin (Cilium)?



DATADOG

Seems related to txq. Let's test a veth with N txq



DATADOG

Summary

- On **recent** clusters we have network issues
- Traffic uses a **single tx queue**
- Traffic uses **single fq_codel qdisc** which triggers drops at high throughputs
- Only impacts **traffic sent through a veth with 1 txqueue** and routed

- Mitigations: **fq_codel => fq** or **multiple tx queues on veth**

DATADOG

How do we debug this?

Where do we look?

Queue selection happens in net device subsystem

(Very) simplified transmit sequence

```
dev_queue_xmit
    __dev_queue_xmit
      netdev_core_pick_tx
        if ndo_select_queue
          ena_select_queue
      __dev_xmit_skb
        __qdisc_run
          qdisc_restart
             sch_direct_xmit
               dev_hard_start_xmit
                 ndo_start_xmit
                 ena_start_xmit
```

Queue selection

Transmission

DATADOG

ena_select_queue

```
static u16 ena_select_queue(struct net_device *dev, struct sk_buff *skb,
                            struct net_device *sb_dev)
{
        u16 qid;
        /* we suspect that this is good for in--kernel network services that
         * want to loop incoming skb rx to tx in normal user generated traffic,
         * most probably we will not get to this
         */
        if (skb_rx_queue_recorded(skb))
                qid = skb_get_rx_queue(skb);
        else
                qid = netdev_pick_tx(dev, skb, NULL);

        return qid;
}
```

This function should compute a flow hash with **skb_tx_hash** and pick a queue
It looks like this is not happening

DATADOG

ena_select_queue

```c
static u16 ena_select_queue(struct net_device *dev, struct sk_buff *skb,
                            struct net_device *sb_dev)
{
        u16 qid;
        /* we suspect that this is good for in--kernel network services that
         * want to loop incoming skb rx to tx in normal user generated traffic,
         * most probably we will not get to this
         */
        if (skb_rx_queue_recorded(skb))
                qid = skb_get_rx_queue(skb);
        else
                qid = netdev_pick_tx(dev, skb, NULL);

        return qid;
}
```

Queue mapping != 0 => Queue has been recorded

```c
static inline bool skb_rx_queue_recorded(const struct sk_buff *skb)
{
        return skb->queue_mapping != 0;
}
```

DATADOG

ena_select_queue

```
static u16 ena_select_queue(struct net_device *dev, struct sk_buff *skb,
                            struct net_device *sb_dev)
{
        u16 qid;
        /* we suspect that this is good for in--kernel network services that
         * want to loop incoming skb rx to tx in normal user generated traffic,
         * most probably we will not get to this
         */
        if (skb_rx_queue_recorded(skb))
                qid = skb_get_rx_queue(skb);
        else
                qid = netdev_pick_tx(dev, skb, NULL);

        return qid;
}
```

Restore the queue value that was recorded

```
static inline u16 skb_get_rx_queue(const struct sk_buff *skb)
{
        return skb->queue_mapping - 1;
}
```

DATADOG

skb->queue_mapping?

Using **bpftrace** to look at the content of queue_mapping

```
kprobe:dev_queue_xmit
{
    $skb = (struct sk_buff *)arg0;
    $skbqm = $skb->queue_mapping;
    $dev = (struct net_device *)$skb->dev;

    $iph = ((struct iphdr *) ($skb->head + $skb->network_header));

    if ($iph->daddr == 168430090) {
       printf("%30s: skb:%20p dev:%3d %20s SKBQM:%7d\n",
              probe, $skb, $dev->ifindex, $dev->name, $skbqm);
    }
}
```

attach kprobe to dev_queue_xmit

retrieve skb data from arg0
get queue_mapping

filter skb to IP 10.10.10.10
print queue_mapping

DATADOG

skb->queue_mapping?

```
$ ip netns exec cni-xxx ping -c 1 10.10.10.10


kprobe:dev_queue_xmit:    dev: 27      eth0     SKBQM:   0
kprobe:dev_queue_xmit:    dev:  3      ens6     SKBQM:   1
```

pod interface

instance interface

different values?

DATADOG

Let's take a more global look

```
     kprobe:dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
   kprobe:__dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
 kprobe:netdev_core_pick_tx   dev: 15                 eth0  SKBQM:  0
 kprobe:dev_hard_start_xmit   dev: 15                 eth0  SKBQM:  0
          kprobe:veth_xmit    dev: 15                 eth0  SKBQM:  0
 kprobe:__netif_receive_skb   dev: 16    lxc9dc9781b7ff1  SKBQM:  1
             kprobe:ip_rcv    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
      kprobe:ip_rcv_finish    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
kprobe:ip_route_input_noref   dev: 16    lxc9dc9781b7ff1  SKBQM:  1
 kprobe:ip_route_input_slow   dev: 16    lxc9dc9781b7ff1  SKBQM:  1
         kprobe:ip_forward    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
     kprobe:dev_queue_xmit    dev:  4                 ens6  SKBQM:  1
   kprobe:__dev_queue_xmit    dev:  4                 ens6  SKBQM:  1
 kprobe:netdev_core_pick_tx   dev:  4                 ens6  SKBQM:  1
   kprobe:ena_select_queue    dev:  4                 ens6  SKBQM:  1
    kprobe:sch_direct_xmit    dev:  4                 ens6  SKBQM:  0
 kprobe:dev_hard_start_xmit   dev:  4                 ens6  SKBQM:  0
     kprobe:ena_start_xmit    dev:  4                 ens6  SKBQM:  0
```

pod netns
Send packet through veth

DATADOG

Let's take a more global look

```
     kprobe:dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
   kprobe:__dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
 kprobe:netdev_core_pick_tx   dev: 15                 eth0  SKBQM:  0
 kprobe:dev_hard_start_xmit   dev: 15                 eth0  SKBQM:  0
          kprobe:veth_xmit    dev: 15                 eth0  SKBQM:  0
 kprobe:__netif_receive_skb   dev: 16    lxc9dc9781b7ff1  SKBQM:  1
            kprobe:ip_rcv     dev: 16    lxc9dc9781b7ff1  SKBQM:  1
      kprobe:ip_rcv_finish    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
kprobe:ip_route_input_noref   dev: 16    lxc9dc9781b7ff1  SKBQM:  1
 kprobe:ip_route_input_slow   dev: 16    lxc9dc9781b7ff1  SKBQM:  1
         kprobe:ip_forward    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
     kprobe:dev_queue_xmit    dev:  4                 ens6  SKBQM:  1
   kprobe:__dev_queue_xmit    dev:  4                 ens6  SKBQM:  1
 kprobe:netdev_core_pick_tx   dev:  4                 ens6  SKBQM:  1
   kprobe:ena_select_queue    dev:  4                 ens6  SKBQM:  1
    kprobe:sch_direct_xmit    dev:  4                 ens6  SKBQM:  0
 kprobe:dev_hard_start_xmit   dev:  4                 ens6  SKBQM:  0
     kprobe:ena_start_xmit    dev:  4                 ens6  SKBQM:  0
```

pod netns
Send packet through veth

host netns
Receive and route

DATADOG

Let's take a more global look

```
     kprobe:dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
   kprobe:__dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
 kprobe:netdev_core_pick_tx   dev: 15                 eth0  SKBQM:  0
 kprobe:dev_hard_start_xmit   dev: 15                 eth0  SKBQM:  0
          kprobe:veth_xmit    dev: 15                 eth0  SKBQM:  0
 kprobe:__netif_receive_skb   dev: 16    lxc9dc9781b7ff1  SKBQM:  1
            kprobe:ip_rcv     dev: 16    lxc9dc9781b7ff1  SKBQM:  1
      kprobe:ip_rcv_finish    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
kprobe:ip_route_input_noref   dev: 16    lxc9dc9781b7ff1  SKBQM:  1
 kprobe:ip_route_input_slow   dev: 16    lxc9dc9781b7ff1  SKBQM:  1
         kprobe:ip_forward    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
     kprobe:dev_queue_xmit    dev:  4                 ens6  SKBQM:  1
   kprobe:__dev_queue_xmit    dev:  4                 ens6  SKBQM:  1
 kprobe:netdev_core_pick_tx   dev:  4                 ens6  SKBQM:  1
   kprobe:ena_select_queue    dev:  4                 ens6  SKBQM:  1
    kprobe:sch_direct_xmit    dev:  4                 ens6  SKBQM:  0
 kprobe:dev_hard_start_xmit   dev:  4                 ens6  SKBQM:  0
     kprobe:ena_start_xmit    dev:  4                 ens6  SKBQM:  0
```

pod netns
Send packet through veth

host netns
Receive and route

host netns
Send packet through ens6

DATADOG

What about skb->queue_mapping?

```
      kprobe:dev_queue_xmit    dev: 15                eth0  SKBQM:  0
    kprobe:__dev_queue_xmit    dev: 15                eth0  SKBQM:  0
 kprobe:netdev_core_pick_tx    dev: 15                eth0  SKBQM:  0
 kprobe:dev_hard_start_xmit    dev: 15                eth0  SKBQM:  0
           kprobe:veth_xmit    dev: 15                eth0  SKBQM:  0
 kprobe:__netif_receive_skb    dev: 16   lxc9dc9781b7ff1  SKBQM:  1
             kprobe:ip_rcv     dev: 16   lxc9dc9781b7ff1  SKBQM:  1
       kprobe:ip_rcv_finish    dev: 16   lxc9dc9781b7ff1  SKBQM:  1
kprobe:ip_route_input_noref    dev: 16   lxc9dc9781b7ff1  SKBQM:  1
 kprobe:ip_route_input_slow    dev: 16   lxc9dc9781b7ff1  SKBQM:  1
          kprobe:ip_forward    dev: 16   lxc9dc9781b7ff1  SKBQM:  1
      kprobe:dev_queue_xmit    dev:  4                ens6  SKBQM:  1
    kprobe:__dev_queue_xmit    dev:  4                ens6  SKBQM:  1
 kprobe:netdev_core_pick_tx    dev:  4                ens6  SKBQM:  1
    kprobe:ena_select_queue    dev:  4                ens6  SKBQM:  1
     kprobe:sch_direct_xmit    dev:  4                ens6  SKBQM:  0
 kprobe:dev_hard_start_xmit    dev:  4                ens6  SKBQM:  0
      kprobe:ena_start_xmit    dev:  4                ens6  SKBQM:  0
```

veth with 1 txq
makes sense

DATADOG

What about skb->queue_mapping?

```
      kprobe:dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
    kprobe:__dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
 kprobe:netdev_core_pick_tx    dev: 15                 eth0  SKBQM:  0
 kprobe:dev_hard_start_xmit    dev: 15                 eth0  SKBQM:  0
           kprobe:veth_xmit    dev: 15                 eth0  SKBQM:  0
 kprobe:__netif_receive_skb    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
             kprobe:ip_rcv     dev: 16    lxc9dc9781b7ff1  SKBQM:  1
       kprobe:ip_rcv_finish    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
kprobe:ip_route_input_noref    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
 kprobe:ip_route_input_slow    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
          kprobe:ip_forward    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
      kprobe:dev_queue_xmit    dev:  4                 ens6  SKBQM:  1
    kprobe:__dev_queue_xmit    dev:  4                 ens6  SKBQM:  1
 kprobe:netdev_core_pick_tx    dev:  4                 ens6  SKBQM:  1
    kprobe:ena_select_queue    dev:  4                 ens6  SKBQM:  1
     kprobe:sch_direct_xmit    dev:  4                 ens6  SKBQM:  0
 kprobe:dev_hard_start_xmit    dev:  4                 ens6  SKBQM:  0
      kprobe:ena_start_xmit    dev:  4                 ens6  SKBQM:  0
```

veth with 1 txq
makes sense

skb_rx_queue_recorded is true
skb_get_rx_queue => 0
(restored)

DATADOG

What about skb->queue_mapping?

```
      kprobe:dev_queue_xmit    dev: 15                eth0  SKBQM:  0
    kprobe:__dev_queue_xmit    dev: 15                eth0  SKBQM:  0
 kprobe:netdev_core_pick_tx    dev: 15                eth0  SKBQM:  0
 kprobe:dev_hard_start_xmit    dev: 15                eth0  SKBQM:  0
           kprobe:veth_xmit    dev: 15                eth0  SKBQM:  0
 kprobe:__netif_receive_skb    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
              kprobe:ip_rcv    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
       kprobe:ip_rcv_finish    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
kprobe:ip_route_input_noref    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
 kprobe:ip_route_input_slow    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
          kprobe:ip_forward    dev: 16    lxc9dc9781b7ff1  SKBQM:  1
      kprobe:dev_queue_xmit    dev:  4                ens6  SKBQM:  1
    kprobe:__dev_queue_xmit    dev:  4                ens6  SKBQM:  1
 kprobe:netdev_core_pick_tx    dev:  4                ens6  SKBQM:  1
    kprobe:ena_select_queue    dev:  4                ens6  SKBQM:  1
     kprobe:sch_direct_xmit    dev:  4                ens6  SKBQM:  0
 kprobe:dev_hard_start_xmit    dev:  4                ens6  SKBQM:  0
      kprobe:ena_start_xmit    dev:  4                ens6  SKBQM:  0
```

Why do we transition to 1 here?

DATADOG

veth_xmit

```
static netdev_tx_t veth_xmit(struct sk_buff *skb, struct net_device *dev)
{
        struct veth_priv *rcv_priv, *priv = netdev_priv(dev);
        struct veth_rq *rq = NULL;
        struct net_device *rcv;
        int length = skb->len;
        bool rcv_xdp = false;
        int rxq;

        rcu_read_lock();
        rcv = rcu_dereference(priv->peer);
        if (unlikely(!rcv)) {
                kfree_skb(skb);
                goto drop;
        }

        rcv_priv = netdev_priv(rcv);
        rxq = skb_get_queue_mapping(skb);
        if (rxq < rcv->real_num_rx_queues) {
                rq = &rcv_priv->rq[rxq];
                rcv_xdp = rcu_access_pointer(rq->xdp_prog);
                skb_record_rx_queue(skb, rxq);
        }
```

It looks like we are storing the queue in the skb
Was this always the case?

DATADOG

veth_xmit

3 drivers/net/veth.c

```
@@ -302,8 +302,7 @@ static netdev_tx_t veth_xmit(struct sk_buff *skb, struct net_device *dev)
302          if (rxq < rcv->real_num_rx_queues) {                  302          if (rxq < rcv->real_num_rx_queues) {
303                  rq = &rcv_priv->rq[rxq];                      303                  rq = &rcv_priv->rq[rxq];
304                  rcv_xdp = rcu_access_pointer(rq->xdp_prog);   304                  rcv_xdp = rcu_access_pointer(rq->xdp_prog);
305 -                if (rcv_xdp)                                  305 +                skb_record_rx_queue(skb, rxq);
306 -                        skb_record_rx_queue(skb, rxq);
307          }                                                     306          }
308                                                                307
309          skb_tx_timestamp(skb);                                308          skb_tx_timestamp(skb);
```

veth: Store queue_mapping independently of XDP prog presence

Currently, veth_xmit() would call the skb_record_rx_queue() only when there is XDP program loaded on peer interface in native mode.

If peer has XDP prog in generic mode, then netif_receive_generic_xdp() has a call to netif_get_rxqueue(skb), so for multi-queue veth it will not be possible to grab a correct rxq.

To fix that, store queue_mapping independently of XDP prog presence on peer interface.

Code is present in

- kernels 5.11.11+
- Longterm kernels 5.10.27+
- Longterm kernels v4.19.184+
- **Ubuntu 20.04.3 kernels 5.11.0-10xx-aws**
- **Ubuntu 20.04.2 kernels 5.8.0-10xx-aws**

DATADOG

How do we fix this?

kernel patch

veth: Do not record rx queue hint in veth_xmit

1 drivers/net/veth.c

```
@@ -335,7 +335,6 @@ static netdev_tx_t veth_xmit(struct sk_buff *skb, struct net_device *dev)
                 */
                use_napi = rcu_access_pointer(rq->napi) &&
                           veth_skb_is_eligible_for_gro(dev, rcv, skb);
-               skb_record_rx_queue(skb, rxq);
        }

        skb_tx_timestamp(skb);
```

Turns out it was a bug and we now have a patch! (many thanks @borkmann)

DATADOG

After the patch

```
      kprobe:dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
    kprobe:__dev_queue_xmit    dev: 15                 eth0  SKBQM:  0
 kprobe:netdev_core_pick_tx    dev: 15                 eth0  SKBQM:  0
 kprobe:dev_hard_start_xmit    dev: 15                 eth0  SKBQM:  0
          kprobe:veth_xmit     dev: 15                 eth0  SKBQM:  0
 kprobe:__netif_receive_skb    dev: 16    lxc9dc9781b7ff1  SKBQM:  0
             kprobe:ip_rcv     dev: 16    lxc9dc9781b7ff1  SKBQM:  0
      kprobe:ip_rcv_finish     dev: 16    lxc9dc9781b7ff1  SKBQM:  0
kprobe:ip_route_input_noref    dev: 16    lxc9dc9781b7ff1  SKBQM:  0
 kprobe:ip_route_input_slow    dev: 16    lxc9dc9781b7ff1  SKBQM:  0
         kprobe:ip_forward     dev: 16    lxc9dc9781b7ff1  SKBQM:  0
      kprobe:dev_queue_xmit    dev:  4                 ens6  SKBQM:  0
    kprobe:__dev_queue_xmit    dev:  4                 ens6  SKBQM:  0
 kprobe:netdev_core_pick_tx    dev:  4                 ens6  SKBQM:  0
    kprobe:ena_select_queue    dev:  4                 ens6  SKBQM:  0
     kprobe:netdev_pick_tx     dev:  4                 ens6  SKBQM:  0
     kprobe:sch_direct_xmit    dev:  4                 ens6  SKBQM:  7
 kprobe:dev_hard_start_xmit    dev:  4                 ens6  SKBQM:  7
      kprobe:ena_start_xmit    dev:  4                 ens6  SKBQM:  7
```

veth_xmit does not store the queue mapping

We now enter netdev_pick_tx
Queue is picked using hash

DATADOG

What about existing nodes?

```
+ /* SPDX-License-Identifier: GPL-2.0 */
+ /* Copyright (C) 2022 Authors of Cilium */
+
+ #ifndef __QM_H_
+ #define __QM_H_
+
+ #include <bpf/ctx/ctx.h>
+
+ static inline void reset_queue_mapping(struct __ctx_buff *ctx __maybe_unused)
+ {
+ #if defined(RESET_QUEUES) && __ctx_is == __ctx_skb
+         /* Workaround for GH-18311 where veth driver might have recorded
+          * veth's RX queue mapping instead of leaving it at 0. This can
+          * cause issues on the phys device where all traffic would only
+          * hit a single TX queue (given veth device had a single one and
+          * mapping was left at 1). Reset so that stack picks a fresh queue.
+          * Kernel fix is at 710ad98c363a ("veth: Do not record rx queue
+          * hint in veth_xmit").
+          */
+         ctx->queue_mapping = 0;
+ #endif
+ }
+
+ #endif /* __QM_H_ */
```

Cilium already loads an eBPF program on veths:

- per packet load-balancing
- network policies

Leverage this program to reset queue mapping

(Requires kernel>=5.1)

We have a mitigation! (many thanks, *again*, @borkmann)

DATADOG

After the eBPF mitigation

```
      kprobe:dev_queue_xmit   dev:  7                eth0 SKBQM:      0
     kprobe:__dev_queue_xmit   dev:  7                eth0 SKBQM:      0
 kprobe:netdev_core_pick_tx   dev:  7                eth0 SKBQM:      0
 kprobe:dev_hard_start_xmit   dev:  7                eth0 SKBQM:      0
           kprobe:veth_xmit   dev:  7                eth0 SKBQM:      0
    kprobe:__dev_forward_skb   dev:  7                eth0 SKBQM:      1
            kprobe:netif_rx   dev:  8    lxc64e6d1d83689 SKBQM:      1
 kprobe:__netif_receive_skb   dev:  8    lxc64e6d1d83689 SKBQM:      1
              kprobe:ip_rcv   dev:  8    lxc64e6d1d83689 SKBQM:      0
       kprobe:ip_rcv_finish   dev:  8    lxc64e6d1d83689 SKBQM:      0
kprobe:ip_route_input_noref   dev:  8    lxc64e6d1d83689 SKBQM:      0
 kprobe:ip_route_input_slow   dev:  8    lxc64e6d1d83689 SKBQM:      0
          kprobe:ip_forward   dev:  8    lxc64e6d1d83689 SKBQM:      0
      kprobe:dev_queue_xmit   dev:  4                ens6 SKBQM:      0
     kprobe:__dev_queue_xmit   dev:  4                ens6 SKBQM:      0
 kprobe:netdev_core_pick_tx   dev:  4                ens6 SKBQM:      0
    kprobe:ena_select_queue   dev:  4                ens6 SKBQM:      0
      kprobe:netdev_pick_tx   dev:  4                ens6 SKBQM:      0
     kprobe:sch_direct_xmit   dev:  4                ens6 SKBQM:      5
 kprobe:dev_hard_start_xmit   dev:  4                ens6 SKBQM:      5
```

Unpatched kernel
=> veth_xmit records queue

Cilium eBPF code on veth resets it to 0

queue is not recorded
=> hash is used to pick queue

DATADOG

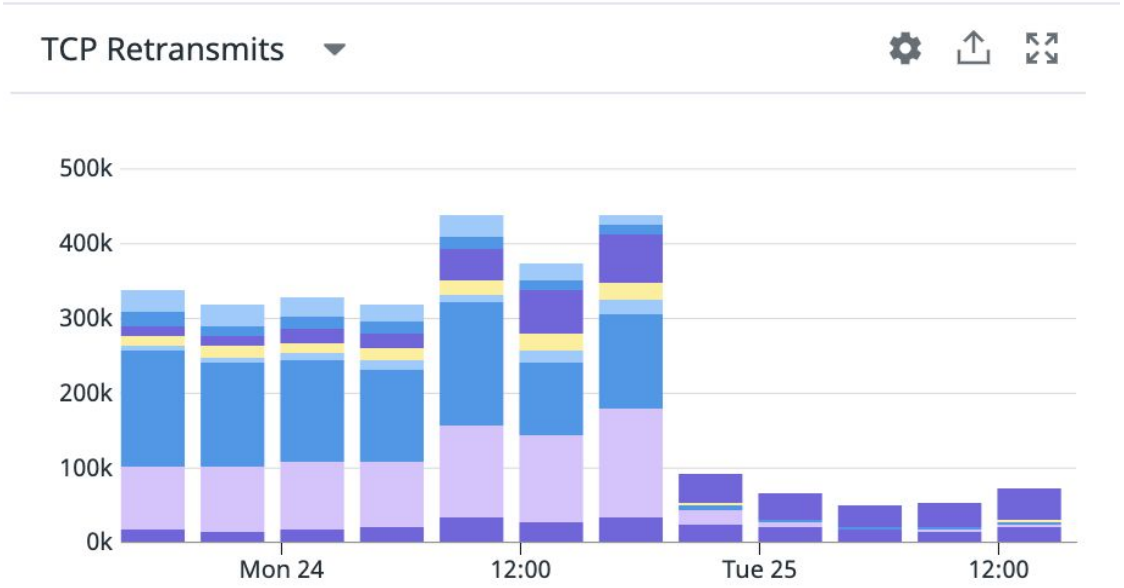
What about our applications?

```
$ ethtool -S ens6 | grep tx_bytes
     queue_0_tx_bytes: 25881756
     queue_1_tx_bytes: 6018503
     queue_2_tx_bytes: 15896451
     queue_3_tx_bytes: 2289972
     queue_4_tx_bytes: 5496205
     queue_5_tx_bytes: 14631267
     queue_6_tx_bytes: 9710508
     queue_7_tx_bytes: 1385099
```

We use all the queues!

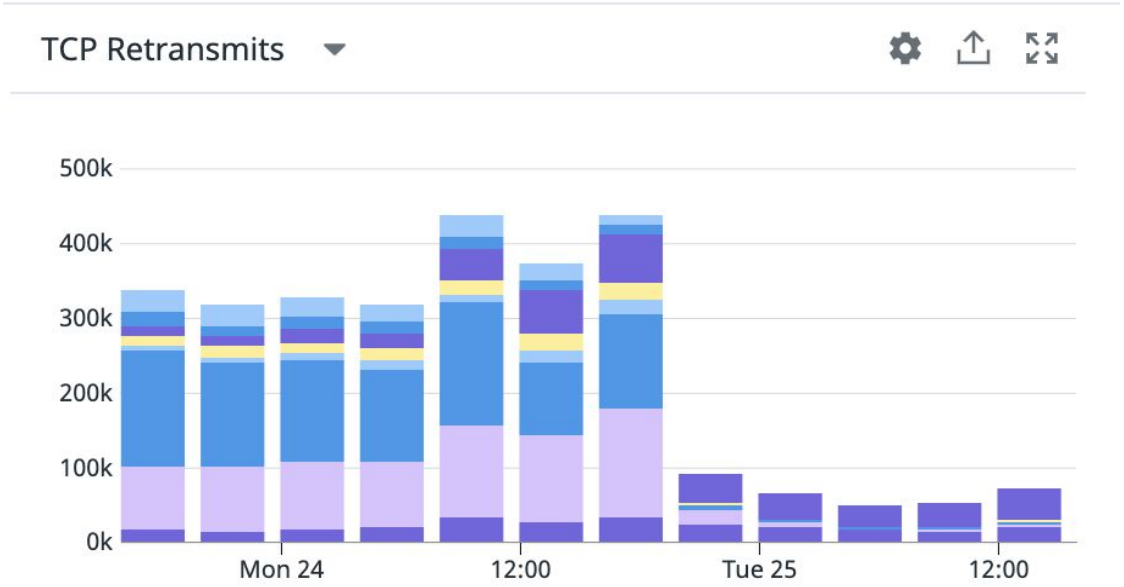
DATADOG

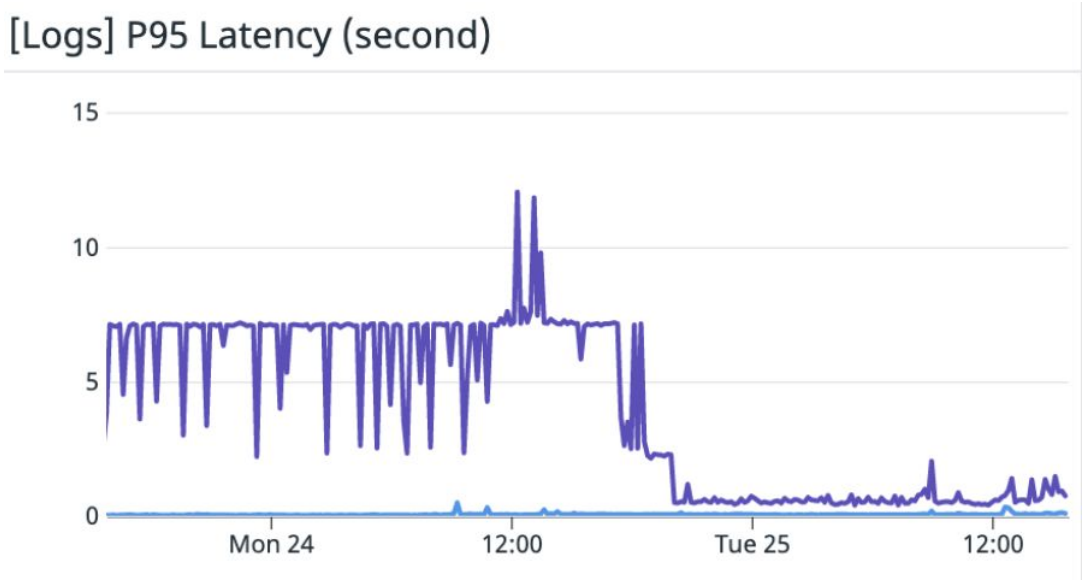
Does it matter?



DATADOG

Does it really matter?





DATADOG

Conclusion

Conclusion

- eBPF makes it possible to debug complex kernel behaviors
  - bpftrace
  - many other tools (pwru)
  - very good way to learn about the kernel
- eBPF makes it possible to patch kernels without an upgrade
- *Sometimes, it is a kernel bug*

Details: https://github.com/cilium/cilium/issues/18311

DATADOG

DATADOG

Thank you!

Questions?