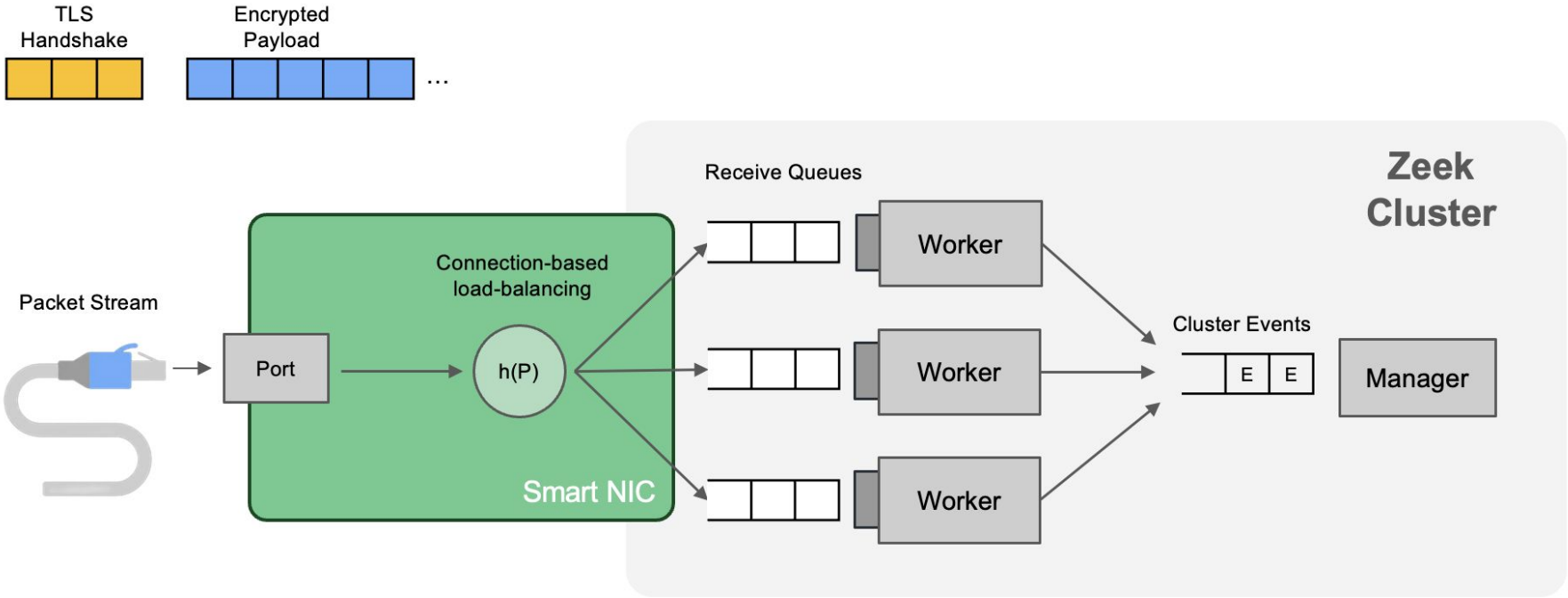
corelight
OPEN SOURCE

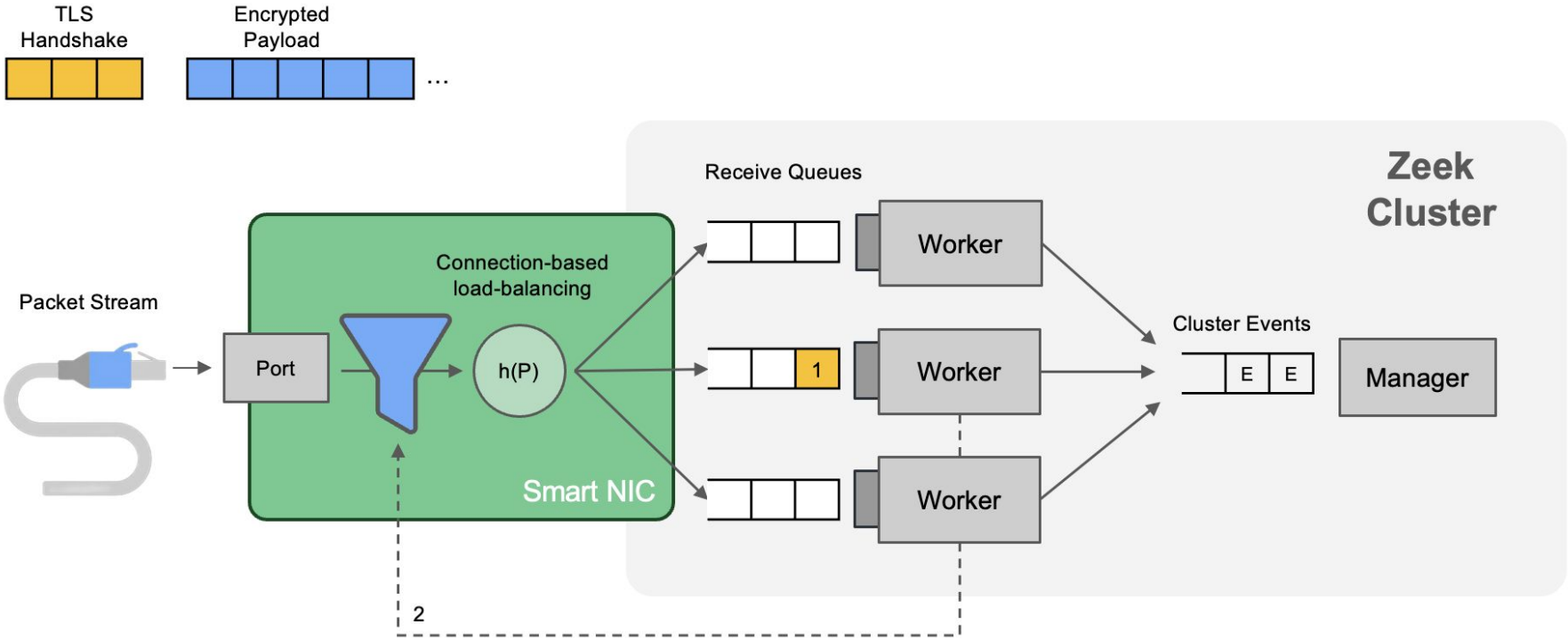
Addressing the Elephant in the Traffic Shunting with Zeek

Evan Typanski & Jan Grashöfer

What's Shunting?



What's Shunting?



Shunting TLS

```
event ssl_established(c: connection)
        {
        do_shunt(c);
        }
```

What does this shunt?

Shunting TLS

```
event ssl_established(c: connection)
        {
        do_shunt(c);
        }
```

What does this shunt?

- Implicit TLS → HTTPS, FTPS, IMAPS, POP3S, SMTPS, XMPPS, LDAPS, …
- Explicit TLS
  - STARTTLS → IMAP, POP3, SMTP, IRC, XMPP
  - Custom → MySQL TLS, PostgreSQL TLS, RDP TLS
- Others → DTLS, TLS in SOCKS

Shunting in conn.log

corelight OPEN SOURCE

Original


```
{
  "ts": 1773850631.852529,
  "uid": "CJKFoj4bpHEhTeaRoj",
  "id.orig_h": "192.168.43.164",
  "id.orig_p": 52651,
  "id.resp_h": "192.0.78.150",
  "id.resp_p": 443,
  "proto": "tcp",
  "service": "ssl",
  "duration": 75.12489604949951,
  "orig_bytes": 2992,
  "resp_bytes": 36372,
  "conn_state": "SF",
  "local_orig": true,
  "local_resp": false,
  "missed_bytes": 0,
  "history": "ShADadfF",
  "orig_pkts": 26,
  "orig_ip_bytes": 4056,
  "resp_pkts": 38,
  "resp_ip_bytes": 37904,
  "ip_proto": 6
}
```


Shunted


```
{
  "ts": 1773850631.852529,
  "uid": "CJKFoj4bpHEhTeaRoj",
  "id.orig_h": "192.168.43.164",
  "id.orig_p": 52651,
  "id.resp_h": "192.0.78.150",
  "id.resp_p": 443,
  "proto": "tcp",
  "service": "ssl",
  "duration": 0.08362483978271484,
  "orig_bytes": 2149,
  "resp_bytes": 1326,
  "conn_state": "S1",
  "local_orig": true,
  "local_resp": false,
  "missed_bytes": 0,
  "history": "ShADad",
  "orig_pkts": 6,
  "orig_ip_bytes": 2413,
  "resp_pkts": 4,
  "resp_ip_bytes": 1498,
  "ip_proto": 6
}
```

Shunting in conn.log

corelight OPEN SOURCE

Original


```
{
  "ts": 1773850631.852529,
  "uid": "CJKFoj4bpHEhTeaRoj",
  "id.orig_h": "192.168.43.164",
  "id.orig_p": 52651,
  "id.resp_h": "192.0.78.150",
  "id.resp_p": 443,
  "proto": "tcp",
  "service": "ssl",
  "duration": 75.12489604949951,
  "orig_bytes": 2992,
  "resp_bytes": 36372,
  "conn_state": "SF",
  "local_orig": true,
  "local_resp": false,
  "missed_bytes": 0,
  "history": "ShADadfF",
  "orig_pkts": 26,
  "orig_ip_bytes": 4056,
  "resp_pkts": 38,
  "resp_ip_bytes": 37904,
  "ip_proto": 6
}
```

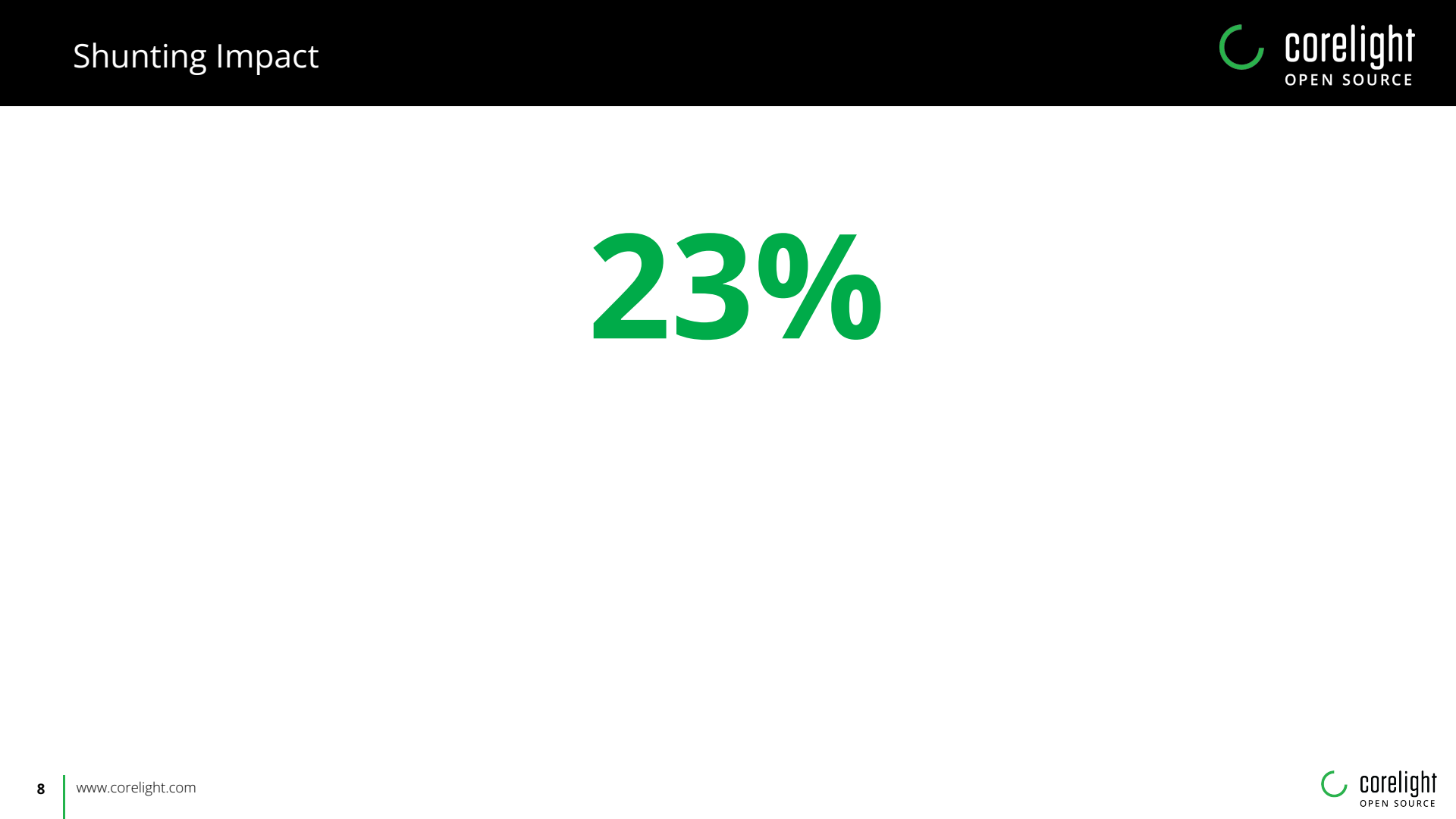

Shunted


```
{
  "ts": 1773850631.852529,
  "uid": "CJKFoj4bpHEhTeaRoj",
  "id.orig_h": "192.168.43.164",
  "id.orig_p": 52651,
  "id.resp_h": "192.0.78.150",
  "id.resp_p": 443,
  "proto": "tcp",
  "service": "ssl",
  "duration": 0.08362483978271484,
  "orig_bytes": 2149,
  "resp_bytes": 1326,
  "conn_state": "S1",
  "local_orig": true,
  "local_resp": false,
  "missed_bytes": 0,
  "history": "ShADad",
  "orig_pkts": 6,
  "orig_ip_bytes": 2413,
  "resp_pkts": 4,
  "resp_ip_bytes": 1498,
  "ip_proto": 6
}
```


Shunted + Metadata


```
{
  "ts": 1773850631.852529,
  "uid": "CJKFoj4bpHEhTeaRoj",
  "id.orig_h": "192.168.43.164",
  "id.orig_p": 52651,
  "id.resp_h": "192.0.78.150",
  "id.resp_p": 443,
  "proto": "tcp",
  "service": "ssl",
  "duration": 75.12345678901234,
  "orig_bytes": 2149,
  "resp_bytes": 1326,
  "conn_state": "S1",
  "local_orig": true,
  "local_resp": false,
  "missed_bytes": 0,
  "history": "ShADad",
  "orig_pkts": 6,
  "orig_ip_bytes": 2413,
  "resp_pkts": 4,
  "resp_ip_bytes": 1498,
  "ip_proto": 6,
  "is_shunted": true,
  "orig_shunted_pkts": 20,
  "orig_shunted_bytes": 1643,
  "resp_shunted_pkts": 34,
  "resp_shunted_bytes": 36406
}
```

Shunting Impact

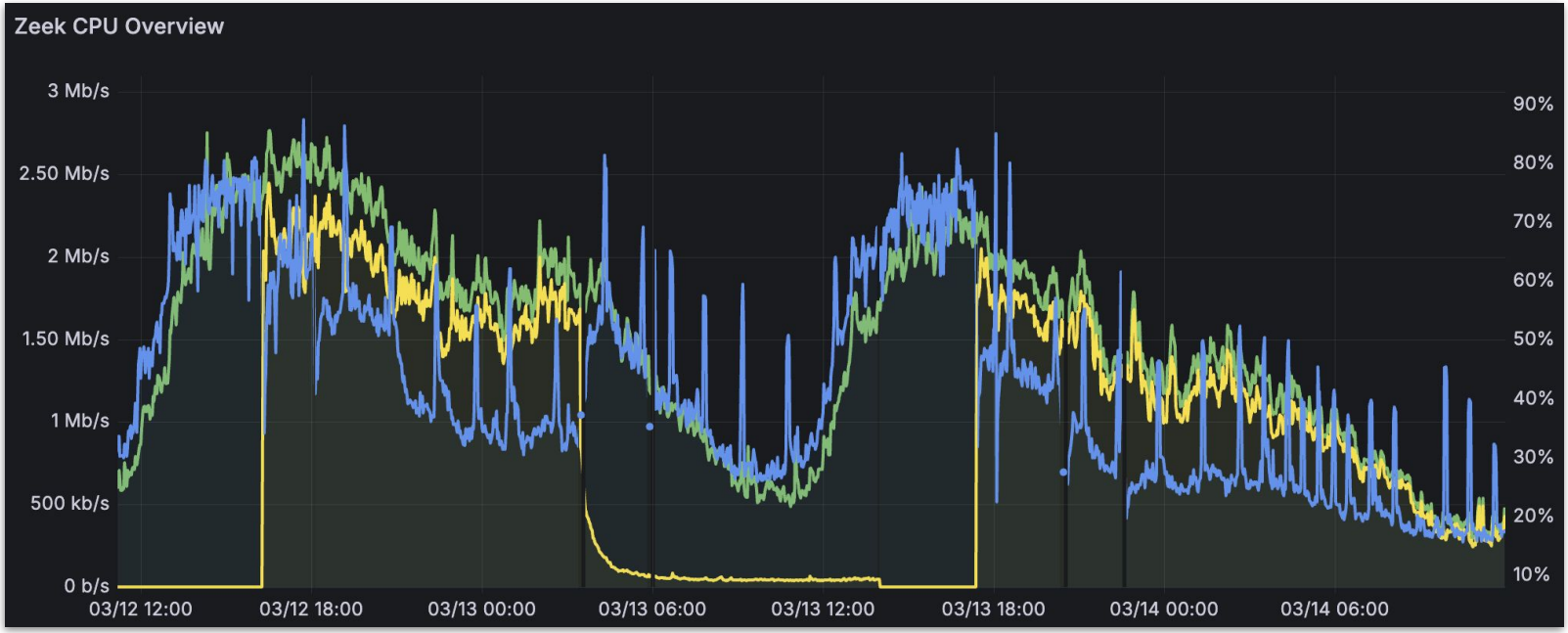


23%

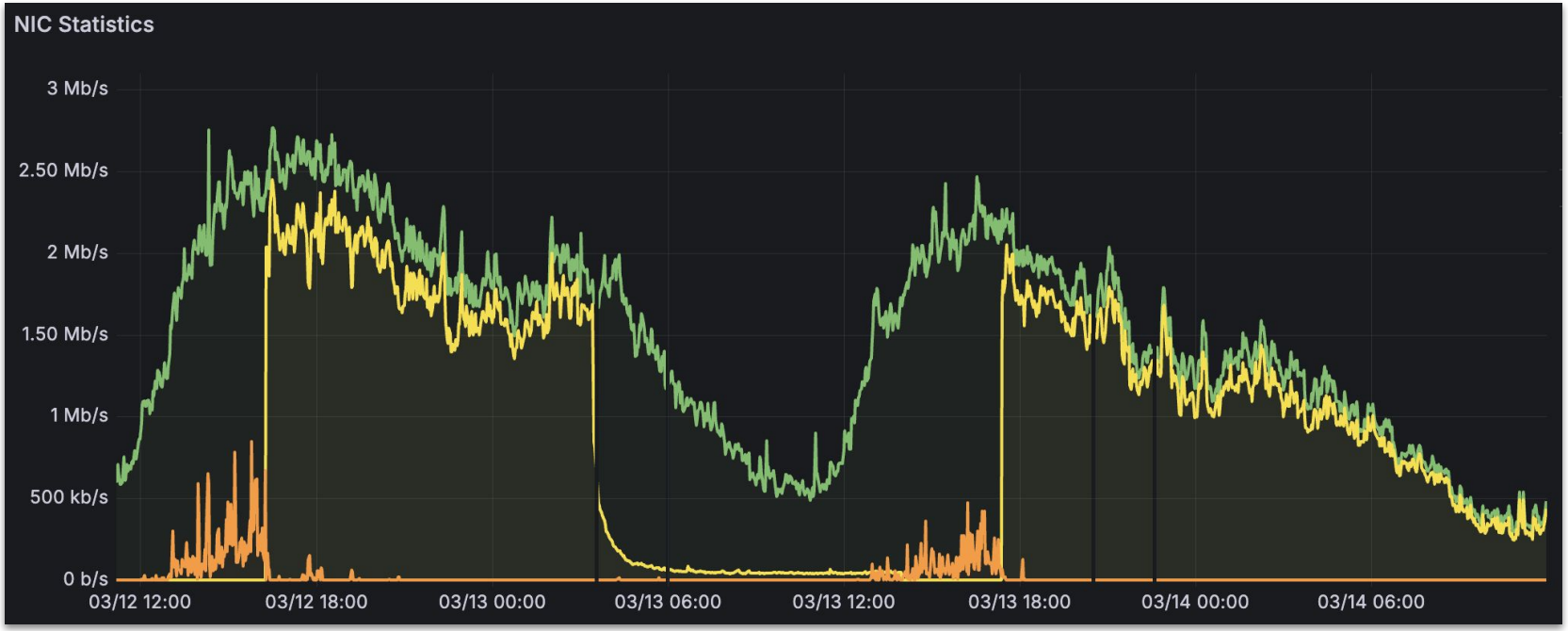
- Lab performance test: Determine max. throughput without packet drops
- Generated traffic mix using ~88% TLS
- Varying configurations: Gains between 7 - 38%

→ **Your mileage ~~may~~ will vary!**

Shunting Impact - Example: CPU Usage



Shunting Impact - Example: Packet Drops



```
event ssl_established(c: connection)
        {
        do_shunt(c);
        }
```

What does this shunt?

- TLS → HTTPS, FTPS, IMAPS, POP3S, SMTPS, XMPPS, LDAPS, …
- STARTTLS → IMAP, POP3, SMTP, IRC, XMPP
- Explicit TLS → MySQL TLS, PostgreSQL TLS, RDP TLS
- Others → DTLS, TLS in SOCKS
- …

What else to shunt?

Shunting Bulk Traffic

corelight OPEN SOURCE

Shunt connections > 1 MB

- for every new connection
  → set threshold per direction

- shunt if crossed

```
option size_threshold = 1000000; # 1MB


event new_connection(c: connection)
    {
    ConnThreshold::set_bytes_threshold(c, size_threshold, T);
    ConnThreshold::set_bytes_threshold(c, size_threshold, F);
    }


event ConnThreshold::bytes_threshold_crossed(c: connection,
    threshold: count, is_orig: bool)
    {
    if ( threshold != size_threshold )
        return;

    do_shunt(c);
    }
```


corelight OPEN SOURCE

Shunting Bulk Traffic

corelight OPEN SOURCE

Shunt connections > 1 MB

- ~~for every new connection~~ 🥵
- for conns > 10 packets
  (excludes short conns like DNS)

  → set threshold per direction

- shunt if crossed

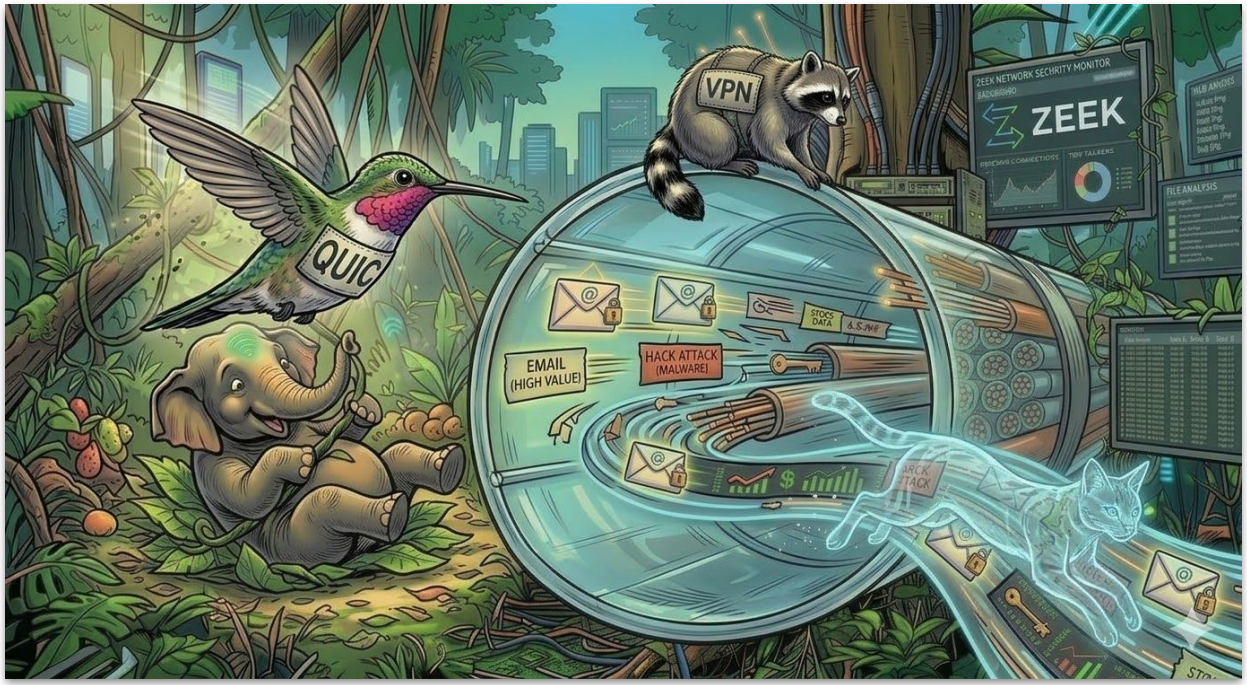
```
option size_threshold = 1000000; # 1MB


const activation_packet_count = 10;
redef ConnThreshold::generic_packet_thresholds
    += {activation_packet_count};

event conn_generic_packet_threshold_crossed(c: connection,
    threshold: count)
    {
    if ( threshold != activation_packet_count )
        return;

    ConnThreshold::set_bytes_threshold(c, size_threshold, T);
    ConnThreshold::set_bytes_threshold(c, size_threshold, F);
    }


event ConnThreshold::bytes_threshold_crossed(c: connection,
    threshold: count, is_orig: bool)
    {
    if ( threshold != size_threshold )
        return;

    do_shunt(c);
    }
```


corelight OPEN SOURCE

Shunting more...

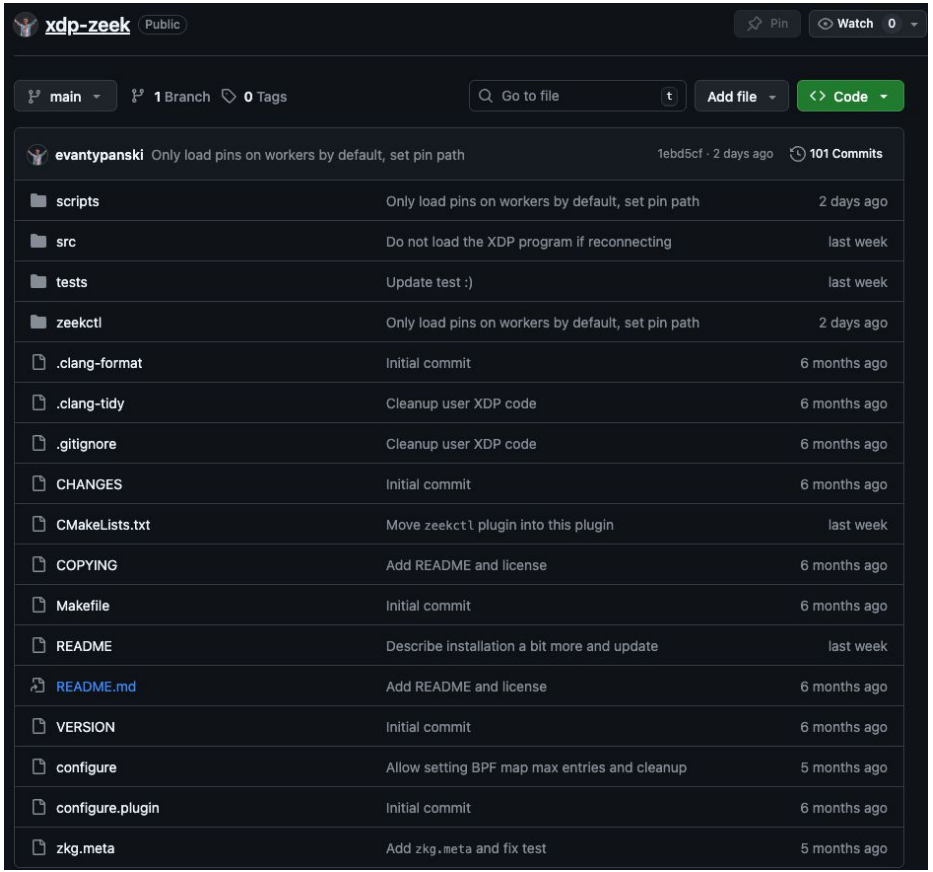
- QUIC
- VPNs (Wireguard, IPSec)
- Unknown protocols
- ...

But how?



Zeek's new XDP shunter

- Active development on a new XDP-based traffic shunter
- Intended to get into Zeek 8.2
- Currently a package: https://github.com/evantypanski/xdp-zeek



What is XDP?

- eXpress Data Path
- eBPF code, runs (before Zeek!) in various places
  - offload mode: on NIC
  - native mode: first thing in the driver
  - SKB or generic mode: after socket buffer allocation (“for testing only”)
- You can access packets before the kernel ever has a chance to see them - very fast and early!

Why shunting?

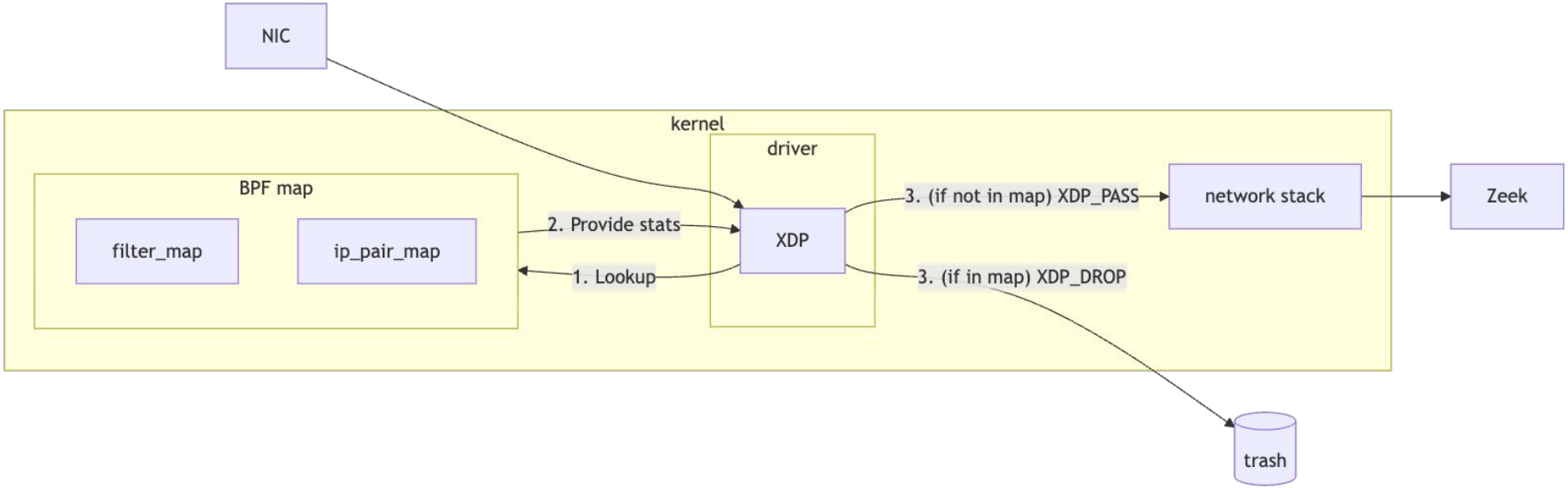
- One of the first XDP examples uses XDP_DROP to drop all packets when it's loaded
- What if we used that to selectively drop packets we don't care about?

```
17  SEC("xdp")
18  int  xdp_pass_func(struct xdp_md *ctx)
19  {
20      return XDP_PASS;
21  }
22
23  SEC("xdp")
24  int  xdp_drop_func(struct xdp_md *ctx)
25  {
26      return XDP_DROP;
27  }
```

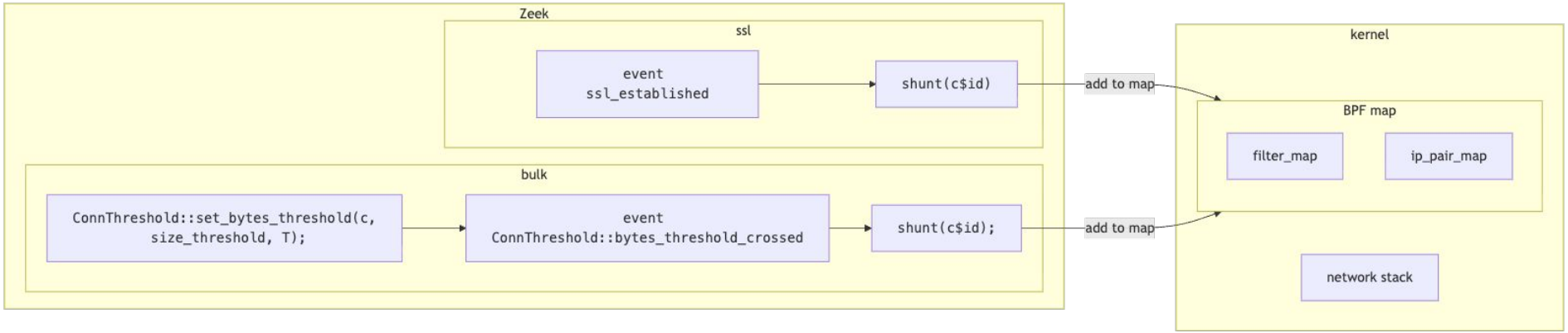
Talking with Zeek: BPF maps

- Zeek has much more information to tell **what** to shunt and **when** to shunt it
- Zeek communicates with XDP via **BPF maps**
- Two ways to shunt: flows (`filter_map`) and IP pairs (`ip_pair_map`)
  - We can add more or change them based on need, but these cannot be dynamic

The flow

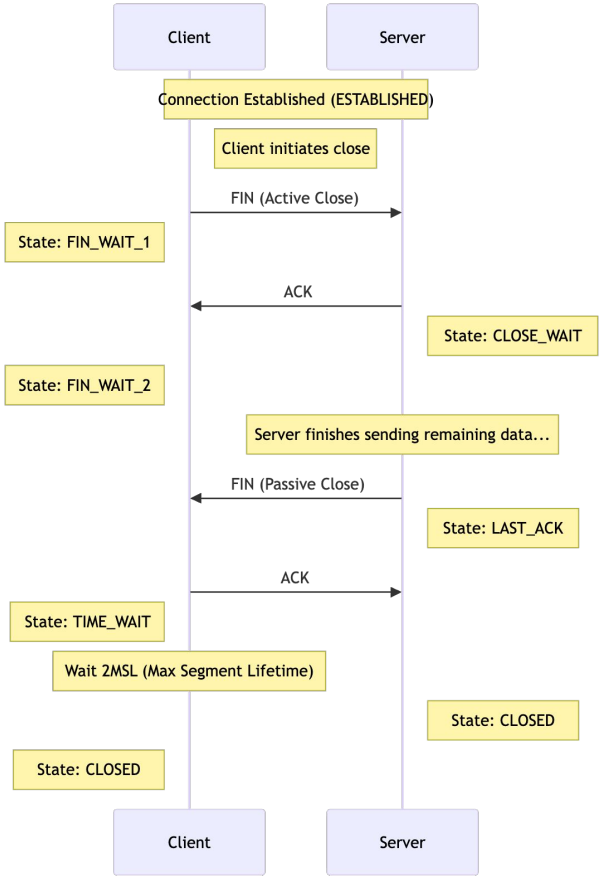

NIC
kernel
driver
BPF map
filter_map
ip_pair_map
2. Provide stats
1. Lookup
XDP
3. (if not in map) XDP_PASS
network stack
Zeek
3. (if in map) XDP_DROP
trash

Zeek's responsibility: Add to map, the flow is shunted!



Unshunting

- Eventually, a connection should be removed from the map
  - otherwise there is unbounded state growth, you will run out of memory
- When do we remove a connection from the map?
- Remember: Zeek no longer sees **any** traffic from the connection, so it cannot react to it
- But if a connection is still getting actively shunted, we want to keep it shunted
- We don't want to unshunt within the XDP program, since it only runs on individual packets, and that hurts configurability for Zeek.



Idea 1: Raise “events” from the XDP program

- XDP still sees the traffic. What if the XDP program told Zeek when a connection finished?
- Done via ringbufs. When a FIN or RST is seen, tell Zeek that the flow is finished via `bpf_ringbuf_submit`

```
struct shunt_val* val = bpf_map_lookup_elem(&filter_map, &tuple);
if ( val ) {
    if ( update_value(val, ctx, from_ip1, fin, rst) == 1 ) {
        struct canonical_tuple* rb_data =
            (struct canonical_tuple*)(bpf_ringbuf_reserve(&filter_rb, sizeof(struct canonical_tuple), 0));
        if ( rb_data ) {
            *rb_data = tuple;
            bpf_ringbuf_submit(rb_data, 0);
        }
    }

    return XDP_DROP;
}
```

Idea 1 cons

- Zeek will have to constantly poll the ring buffer to see if a flow should be unshunted
- Hard for XDP to tell if we actually closed the connection
- Only helps for TCP connections that ended gracefully

… but it’s cool 😎

Idea 2: Forward TCP control packets

- Zeek uses TCP packets like FIN and RST to tell if a connection is finished
- We can just use XDP_PASS to forward those to Zeek, so Zeek can then make the decision!
- Easy implementation:

```
136         // Forward all TCP control packets
137         is_control_packet = tcph->fin || tcph->rst || tcph->syn || tcph->ack;
138
139         if ( is_control_packet )
140             return XDP_PASS;
```

Idea 2 cons

- Some TCP control packets (ACK in particular) can carry data. This means for connections like SSH, hardly anything will get shunted!
- Even if we only forward “pure acks” (without data), those aren’t common enough for Zeek to reliably know to close the connection
- (Still) Only helps for TCP connections that ended gracefully

ssh && tcp.flags.ack == 1

No.	Length	Info	Flags
4	93	Server: Protocol (SSH-1.99-OpenSSH_3.9p1)	·····AP···
6	90	Client: Protocol (SSH-2.0-OpenSSH_3.8.1p1)	·····AP···
8	674	Client: Key Exchange Init	·····AP···
9	710	Server: Key Exchange Init	·····AP···
12	90	Client: Diffie-Hellman Group Exchange Request	·····AP···
14	222	Server: Diffie-Hellman Group Exchange Group	·····AP···
16	210	Client: Diffie-Hellman Group Exchange Init	·····AP···
17	534	Server: Diffie-Hellman Group Exchange Reply, New Keys	·····AP···
19	82	Client: New Keys	·····AP···
21	114	Client: Encrypted packet (len=48)	·····AP···
23	118	Server: Encrypted packet (len=48)	·····AP···
25	130	Client: Encrypted packet (len=64)	·····AP···
26	150	Server: Encrypted packet (len=80)	·····AP···
28	162	Client: Encrypted packet (len=96)	·····AP···
29	118	Server: Encrypted packet (len=48)	·····AP···
31	162	Client: Encrypted packet (len=96)	·····AP···
32	150	Server: Encrypted packet (len=80)	·····AP···
34	594	Client: Encrypted packet (len=528)	·····AP···
36	550	Server: Encrypted packet (len=480)	·····AP···
38	642	Client: Encrypted packet (len=576)	·····AP···
40	102	Server: Encrypted packet (len=32)	·····AP···
42	130	Client: Encrypted packet (len=64)	·····AP···
43	118	Server: Encrypted packet (len=48)	·····AP···
45	514	Client: Encrypted packet (len=448)	·····AP···

Idea 3 (current): Zeek times out in the same way as before

- Zeek *already* has a timer to time out connections. It's just not as reliable when Zeek doesn't see packets
  - We could use these for a generic "garbage collection" but that ends up with a lot more issues
- What if we could inform Zeek that a connection should not be removed, even though it "should" be timed out?
- New feature! The `connection_timing_out` hook (PR #5223) to tell Zeek not to remove the connection

```
hook ::connection_timing_out(c: connection)
    {
    if ( ! shunt_timeout )
        return;

    local stats = XDP::Shunt::ConnID::shunt_stats(c);

    # Early abort for connections that aren't shunted.
    if ( ! stats?$present )
        return;

    if ( stats?$timestamp
        && network_time() - stats$timestamp < shunted_inactivity_timeout )
        break;
    }
```

Idea 3 cons

- Zeek will show the shunted connection timed out, even if it closed gracefully
- Even lively connections get checked each timeout interval, could be wasteful
- If a worker crashes, it loses track of the connection, so it’s never unshunted

Current Issues with the XDP shunting (it's still in development!)

- Still in a package, not in Zeek's core (https://github.com/evantypanski/xdp-zeek)
- Clusters are annoying
  - Loading XDP programs is a privileged operation, so may need root for it
  - XDP is per-interface, need to clean up if multiple workers are on one interface (eg af_packet load balancing)
- `xdp-loader` is insufficient
  - It's the classic way to load XDP programs, but it's too generic for some things (configuring map sizes)
  - But, it allows loading multiple XDP programs with a "dispatcher"
  - Working on a custom loader with extra functionality for Zeek in it (eg analyzing the current map, unshunting connections from crashed workers)
- Frags
  - XDP lets you work with large MTU sizes, but only some drivers have support
  - If you enable jumbo frames, XDP may be tricky (or impossible) at the moment

How can YOU use it?

- Need to run on Zeek `master` branch (for the `connection_timing_out` hook)
- Then install the package https://github.com/evantypanski/xdp-zeek
- Enable XDP in zeekctl (`zeekctl.cfg`):

  xdp.enabled = True

- That will load the XDP program in your cluster with `zeekctl deploy` (make sure you're root!)
- Then load the shunting policy scripts for some basic shunting (in `local.zeek`):

```
@load xdp/shunt/policy
```

corelight OPEN SOURCE

Shunting Pitfalls in the Field

- Traffic mix → YMWV 😉
- Consider shunting metadata
- Event selection

jan.grashoefer@corelight.com
github.com/J-Gras
linkedin.com/in/jan-grashoefer
@jan

XDP Shunting

- Available at github.com/evantypanski/xdp-zeek
- Unshunting is hard!
- Let Evan know if anything explodes :)

etyp@zeek.org
github.com/evantypanski
linkedin.com/in/evan-typanski
@evantypanski