CLOUDFLARE

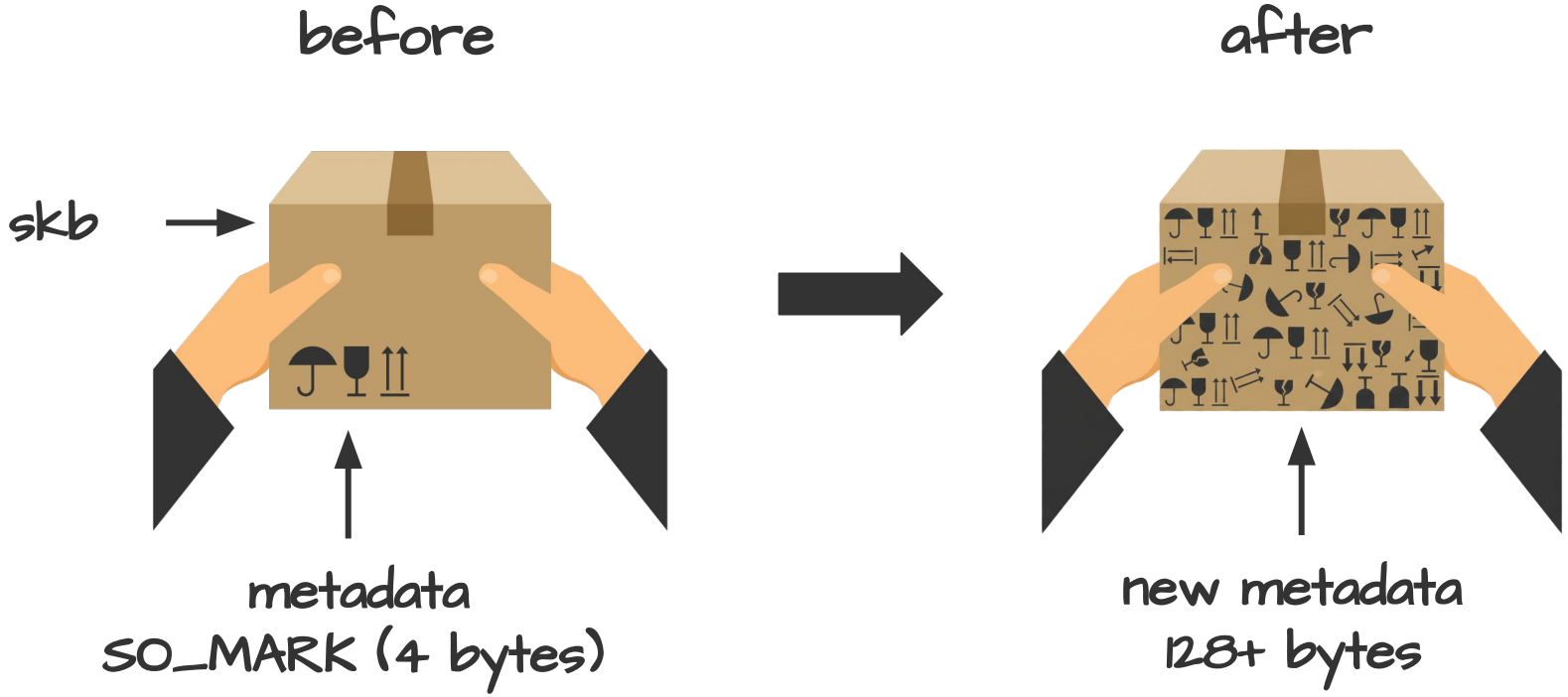
Thrice the charm: an skb extension for BPF metadata

Jakub Sitnicki
Performance Engineer
Cloudflare

Andrej Stender
Systems Engineer
Cloudflare

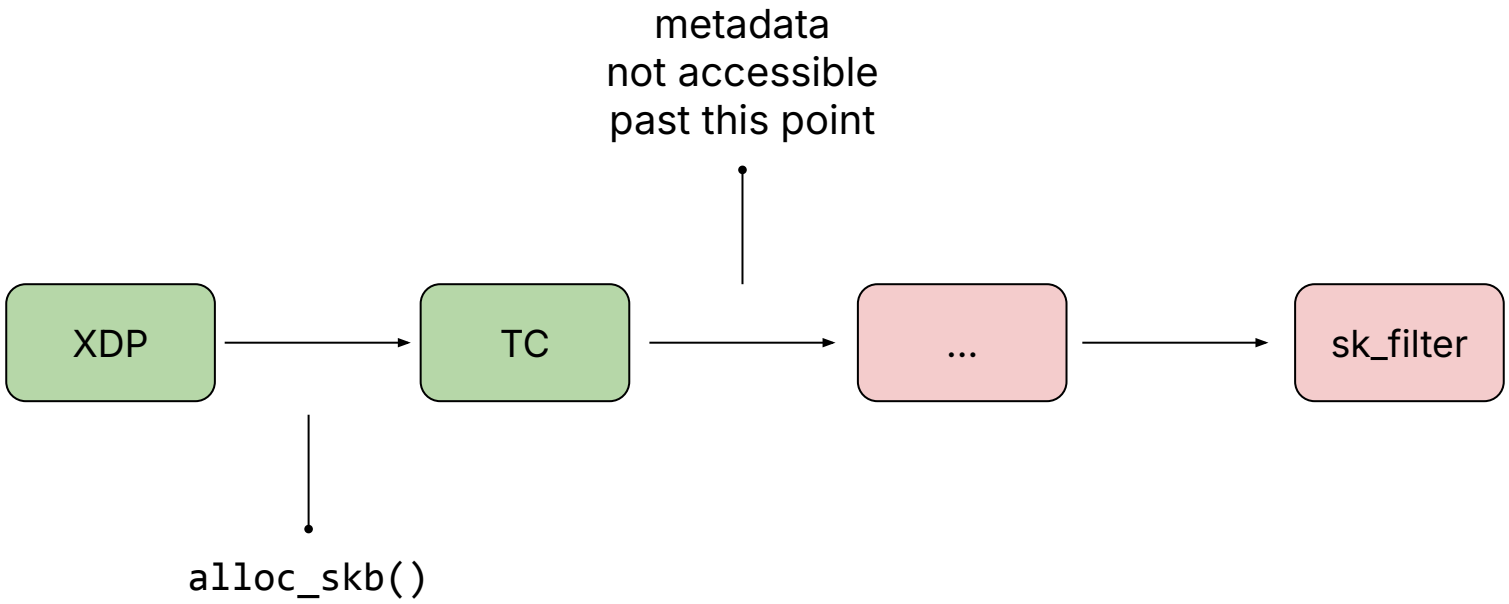
Netdev 0x1A
July 2026
Rome, Italy

Motivation – Support tens of bytes of packet metadata

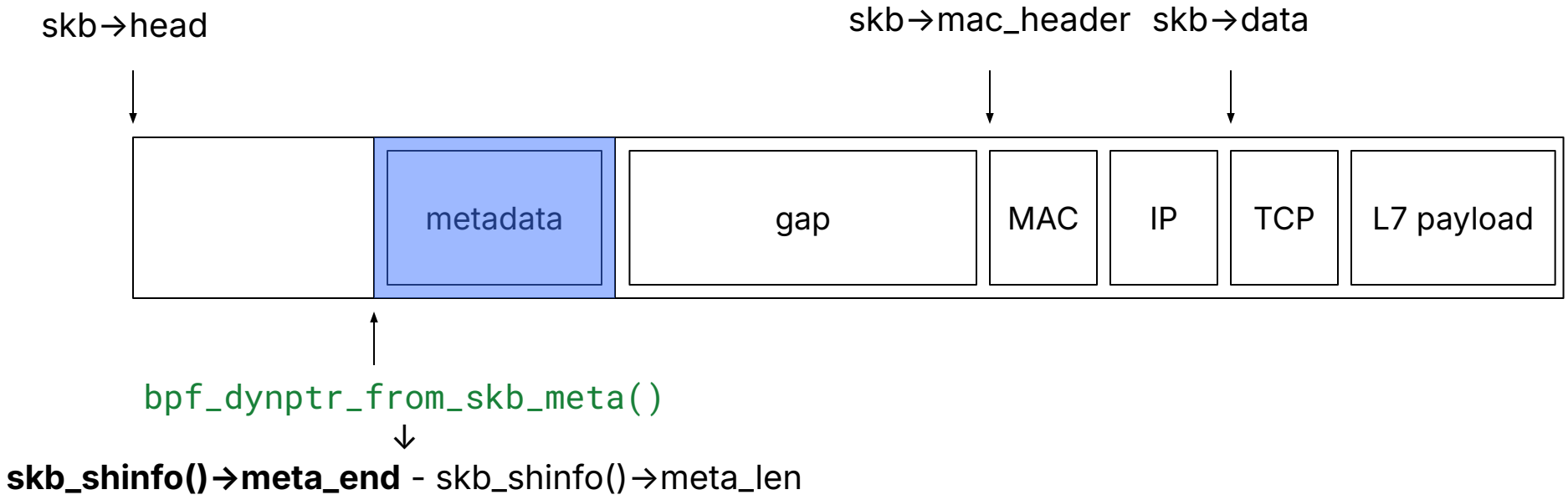



First time
was a miss...

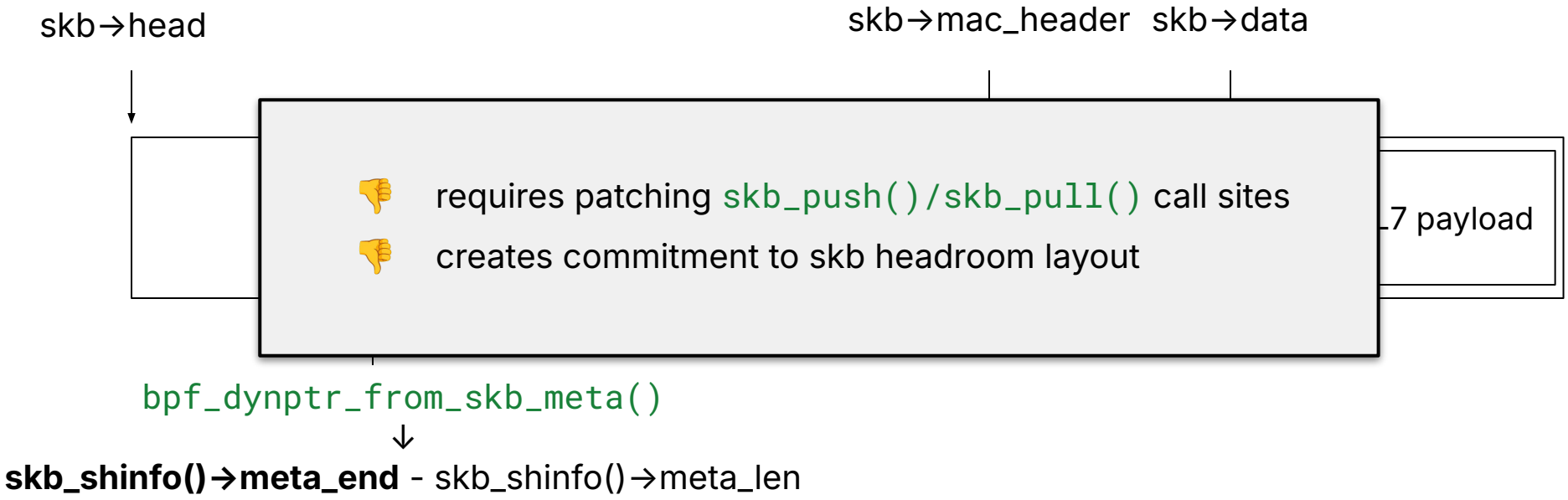
XDP/skb metadata



Proposed layout –
Metadata anywhere within the headroom

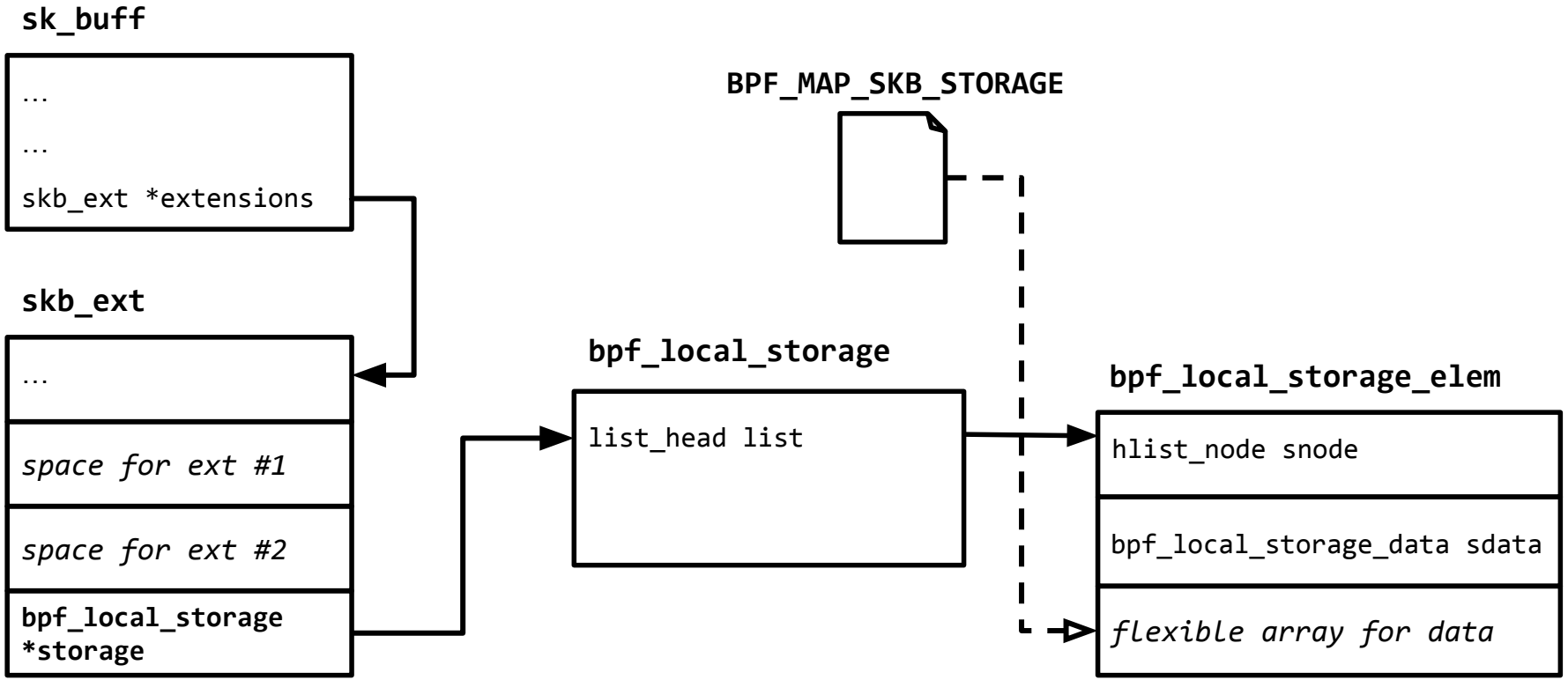


Proposed layout –
Metadata anywhere within the headroom

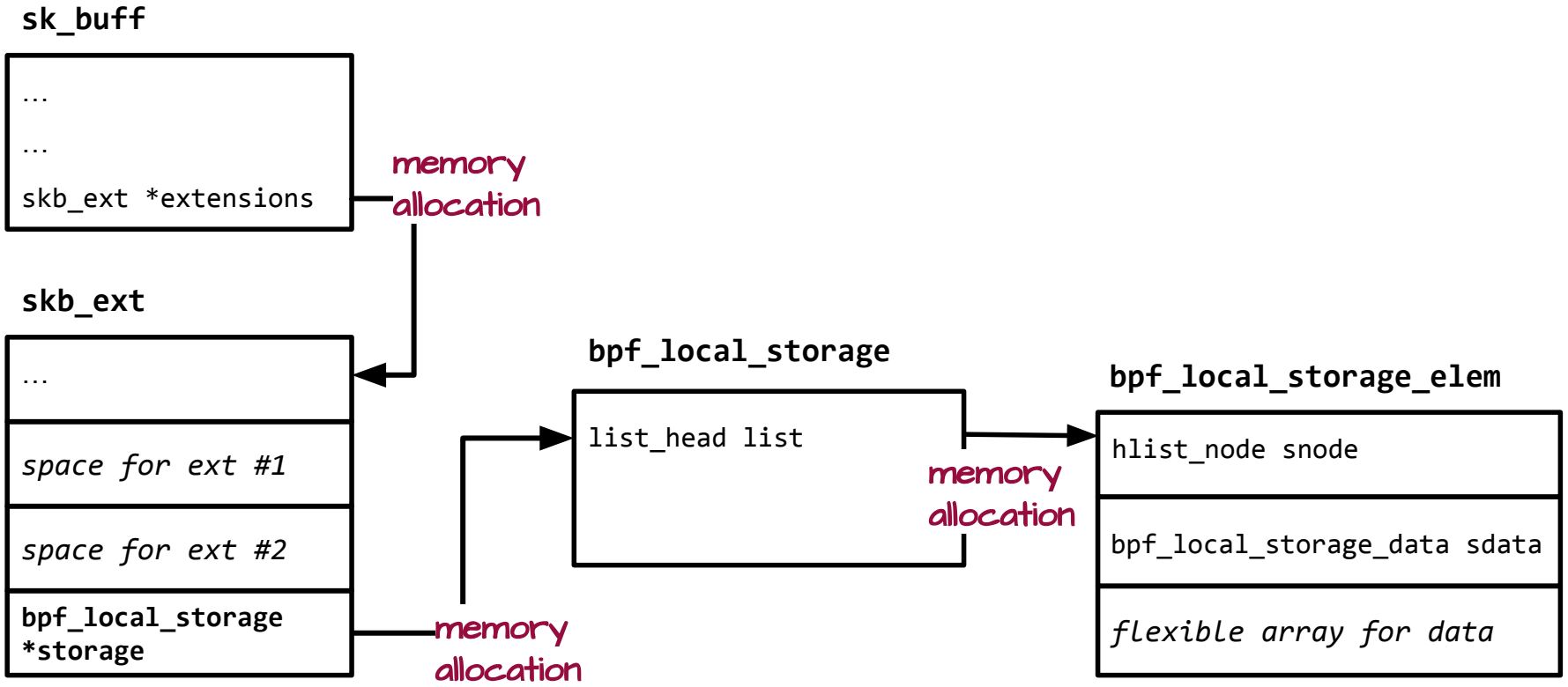



Second time
also a miss...

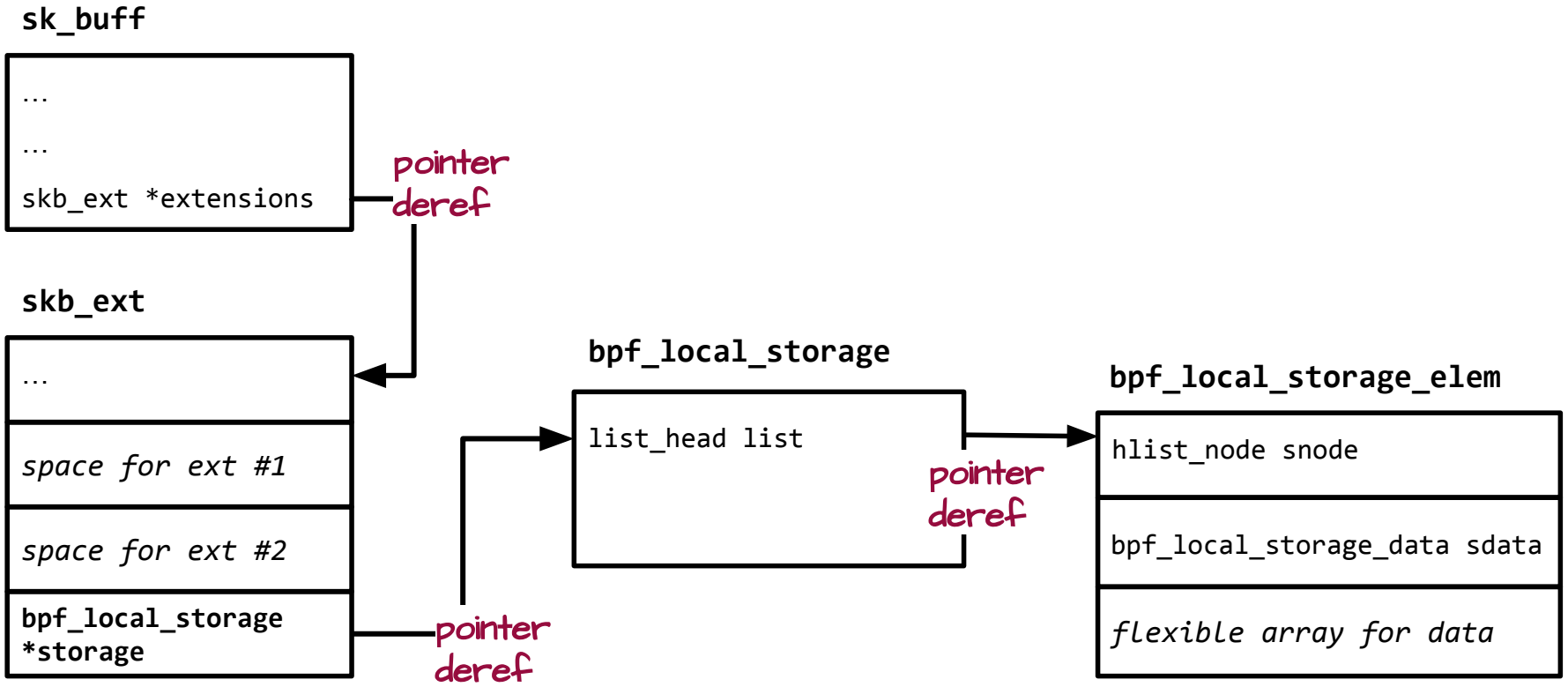
skb extension for BPF local storage



skb extension for BPF local storage

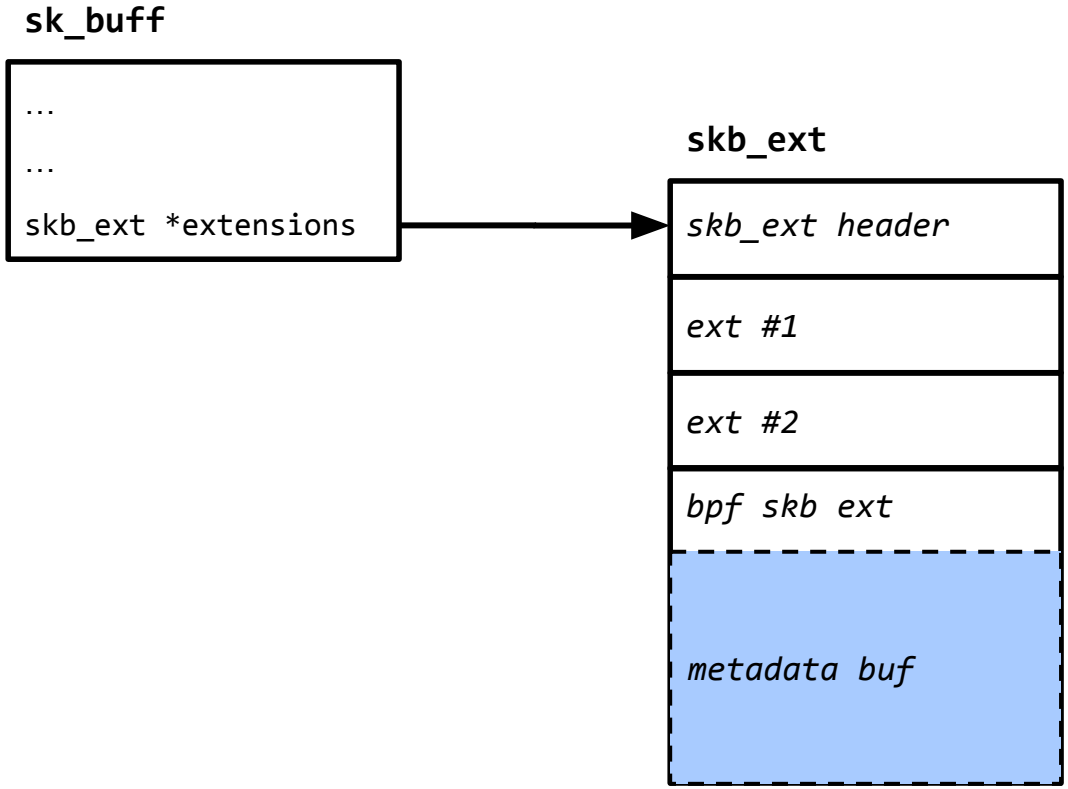


skb extension for BPF local storage




Third time’s
the charm?

Keep the skb extension but embed the metadata



skb extension for BPF

CLOUDFLARE

```
#ifdef CONFIG_BPF_SKB_EXT

struct bpf_skb_ext {
    u64 flags;
    u8 buf[CONFIG_BPF_SKB_EXT_SIZE] __aligned(8);
};

#endif /* CONFIG_BPF_SKB_EXT */
```

skb extension for BPF

CLOUDFLARE

```
config BPF_SKB_EXT
    bool "skb extension for BPF metadata"
    depends on BPF_SYSCALL
    select SKB_EXTENSIONS

config BPF_SKB_EXT_SIZE
    int "Size of the BPF skb extension metadata buffer"
    depends on BPF_SKB_EXT
    range 1 256
    default 64
```

BPF access via dynptr

```
int bpf_dynptr_from_skb_ext(skb, size, flags, &ptr)
```

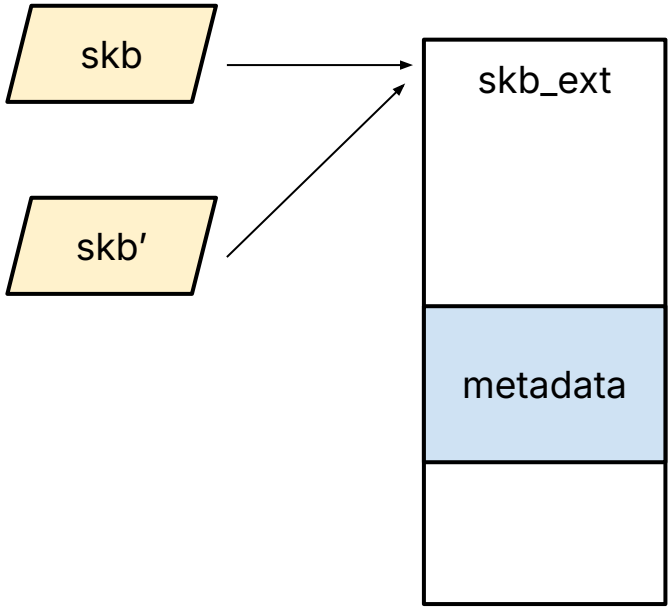
- `skb` - pointer to `sk_buff` or `__sk_buff`
- `size` - `0` for max (=`CONFIG_BPF_SKB_EXT_SIZE`)
- flags:
  - `F_CREATE` - pass to allocate/COW (read-write), `0` to find (read-only)
  - `F_NO_SCRUB` - keep the extension across `skb_scrub_packet()`

BPF access via dynptr

```
int bpf_dynptr_from_skb_ext(skb, size, flags, &ptr)
```

- return values:
  - 0 - dynptr ready to use
  - -ENOENT - extension not found (when not creating)
  - -ENOMEM - allocation failed
  - -EINVAL - invalid flags
  - -E2BIG - size exceeds CONFIG_BPF_SKB_EXT_SIZE

Access for skb clones - read-only by default

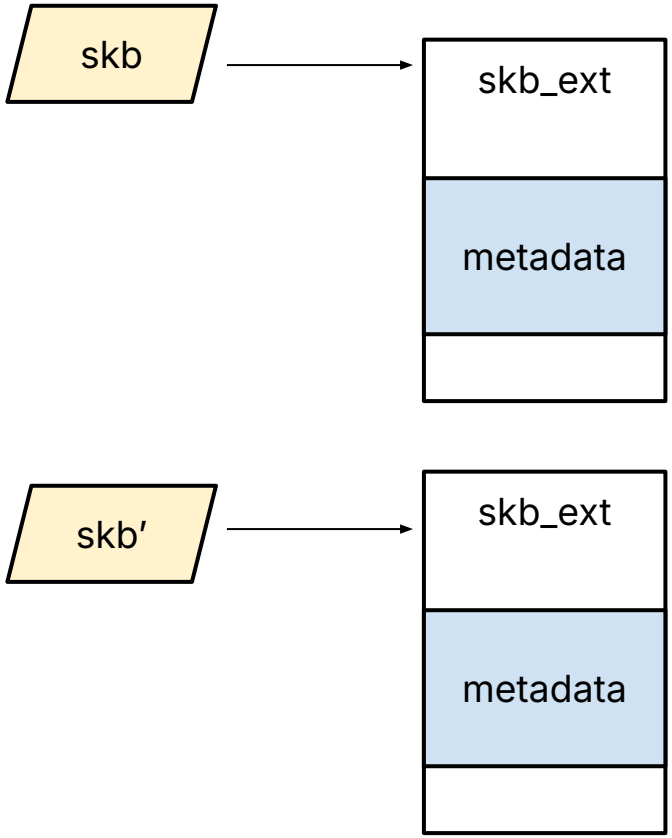



```
bpf_dynptr_from_skb_ext(skb', …,
                        0 /* no flags */, …)
```

READ-ONLY DYNPTR

(uses `skb_ext_find()` underneath)

Access for skb clones - read-write / COW on demand

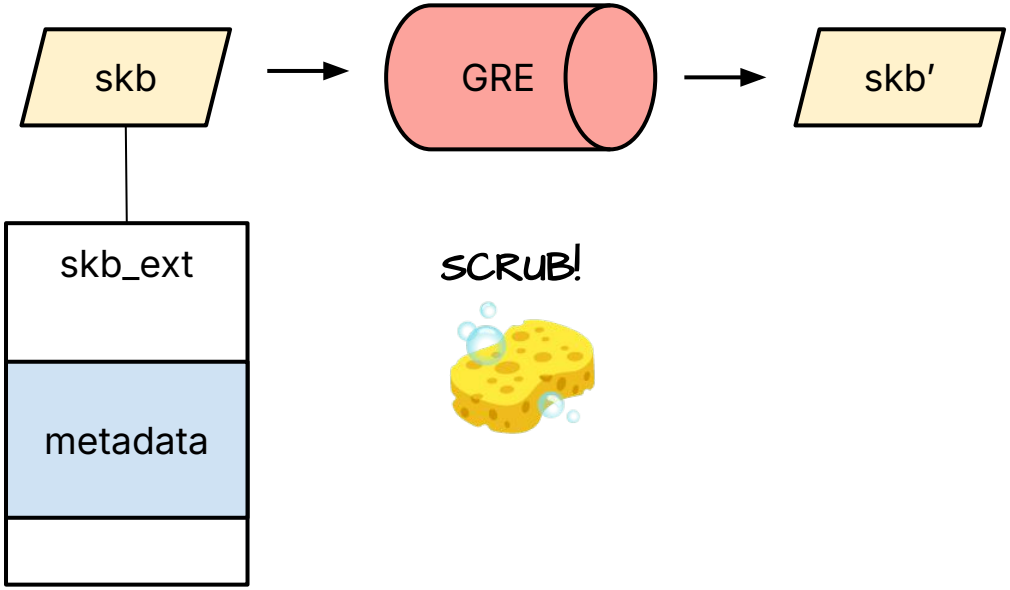



```
bpf_dynptr_from_skb_ext(skb', …,
                        F_CREATE, …)
```

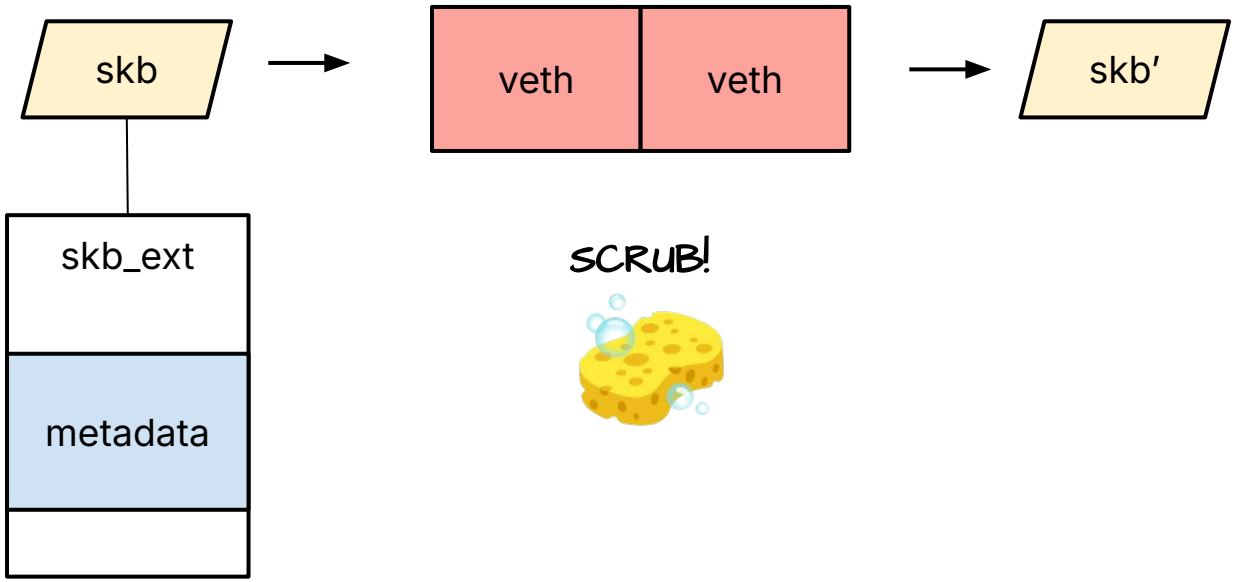
READ-WRITE DYNPTR

(uses `skb_ext_add()` underneath to COW)

skb scrubbing on tunnel encap/decap

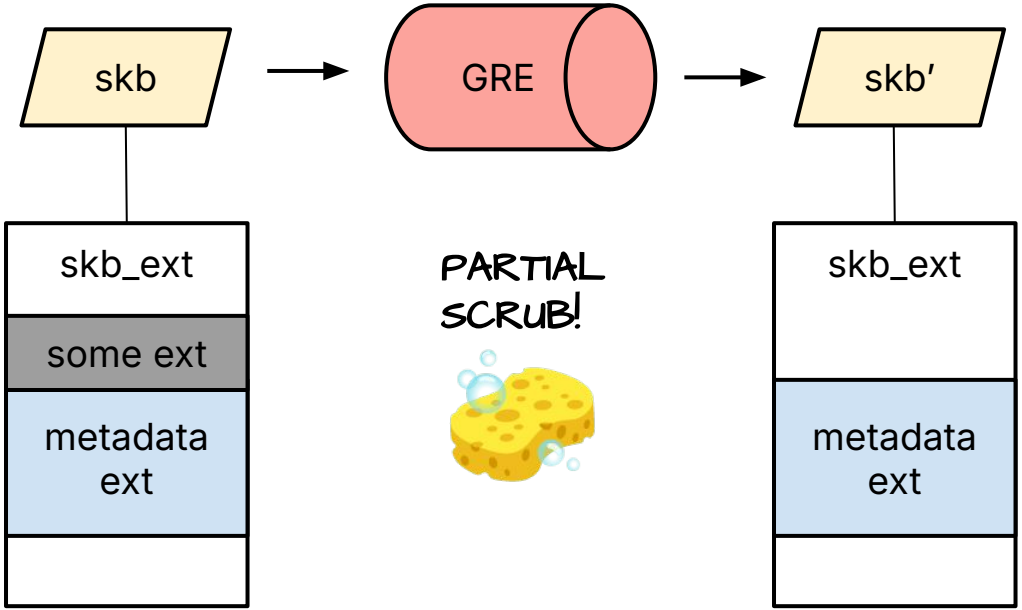


skb scrubbing when crossing netns boundary



RFC: let skb ext's opt out of scrubbing

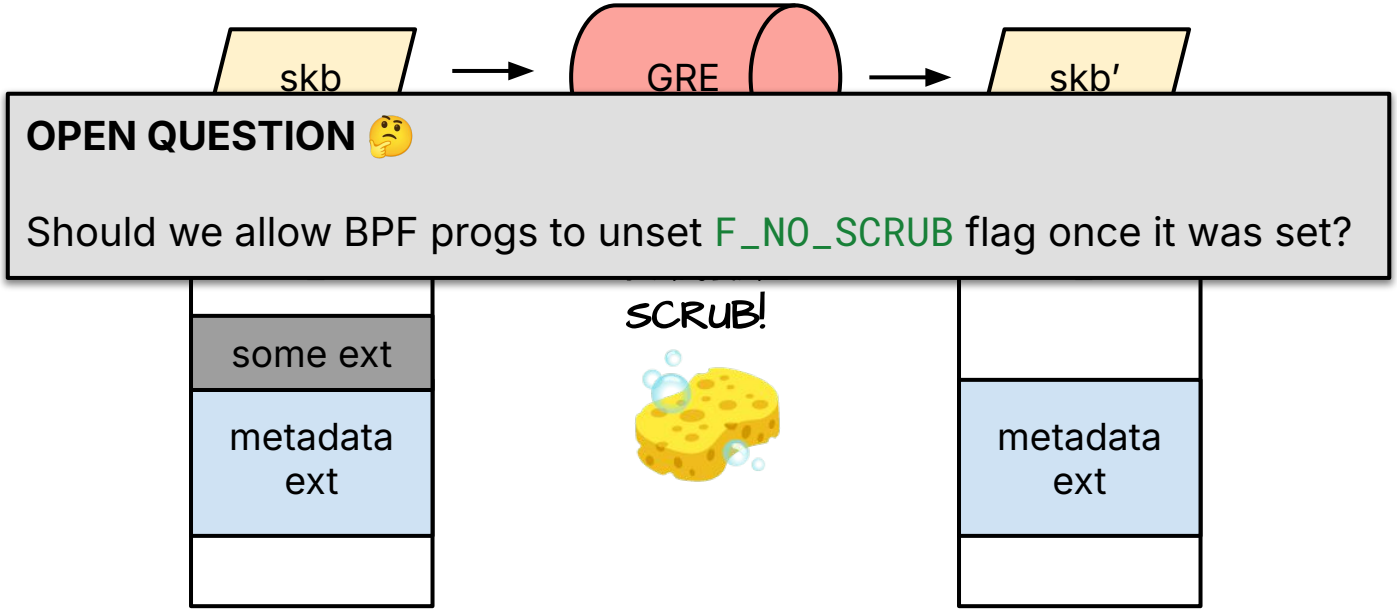
```
bpf_dynptr_from_skb_ext(skb, …, F_CREATE | F_NO_SCRUB, …)
```



Requires changes to skb extensions framework

RFC: let skb ext's opt out of scrubbing

```
bpf_dynptr_from_skb_ext(skb, …, F_CREATE | F_NO_SCRUB, …)
```



Requires changes to skb extensions framework

`skb_ext` size considerations

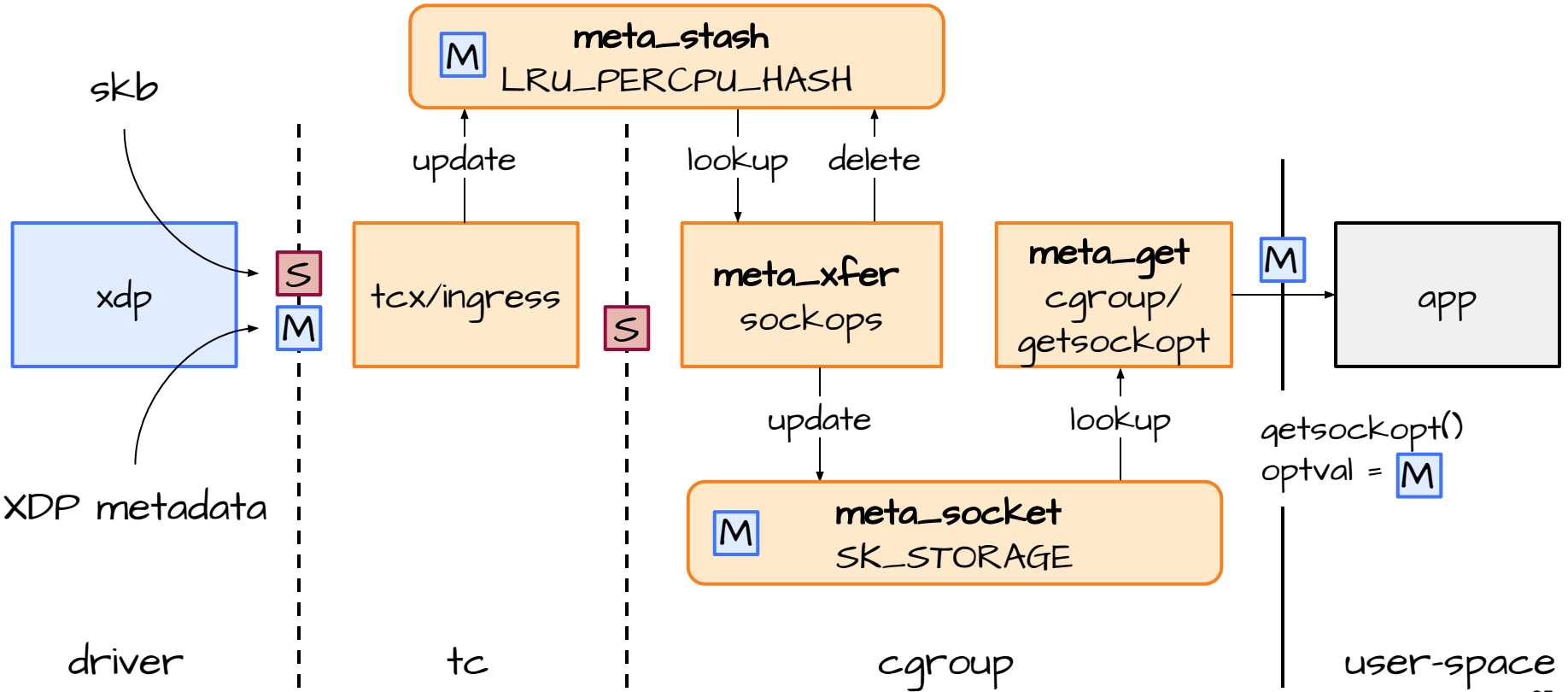
```
923m16$ sudo cat /sys/kernel/slab/skbuff_ext_cache/object_size
448
```

448B in total:

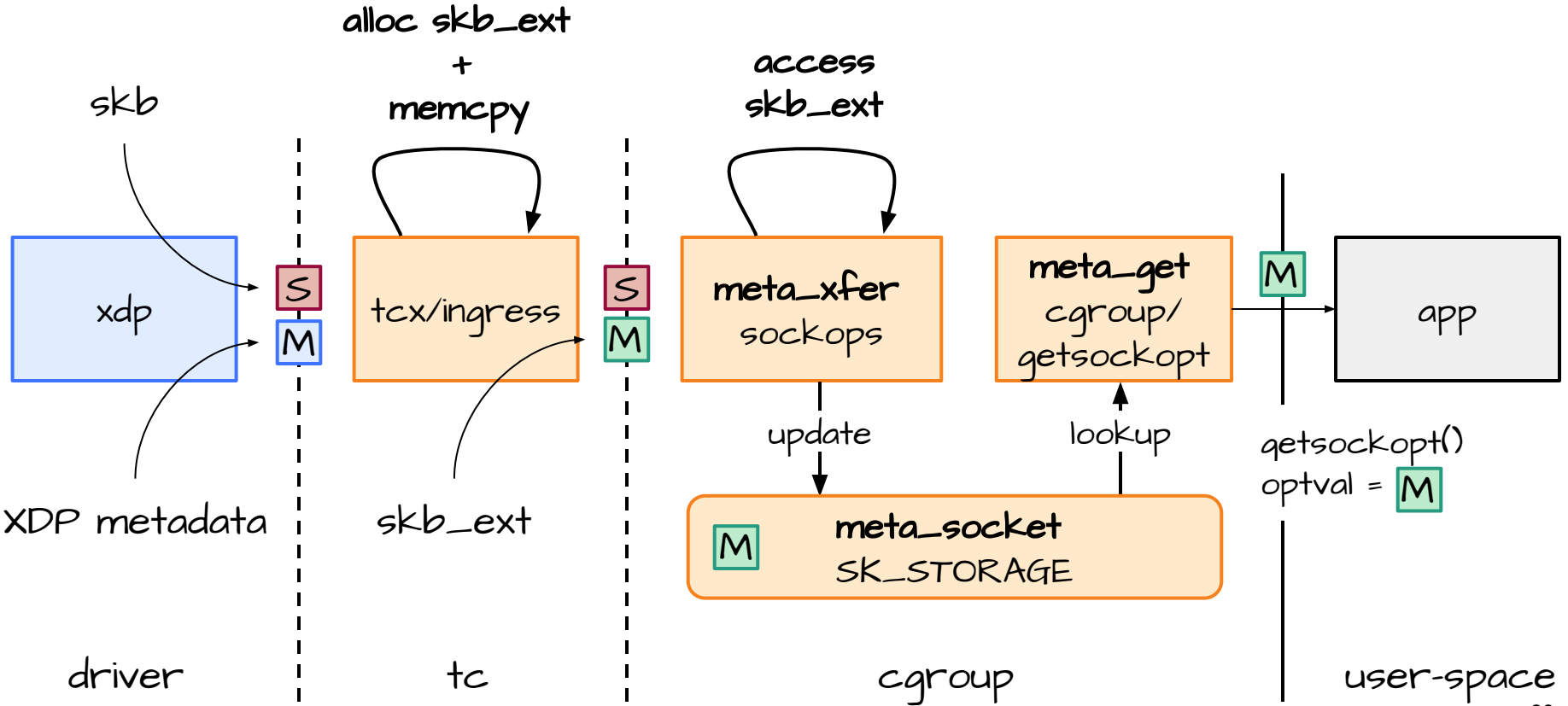
- 16B for `skb_ext` header
- 256B for `bpf_skb_ext` – tweak with CONFIG_BPF_SKB_EXT_SIZE
- 176B for other extensions
  - 24B for `XFRM` – some state used only for HW offload?

Experiment time!

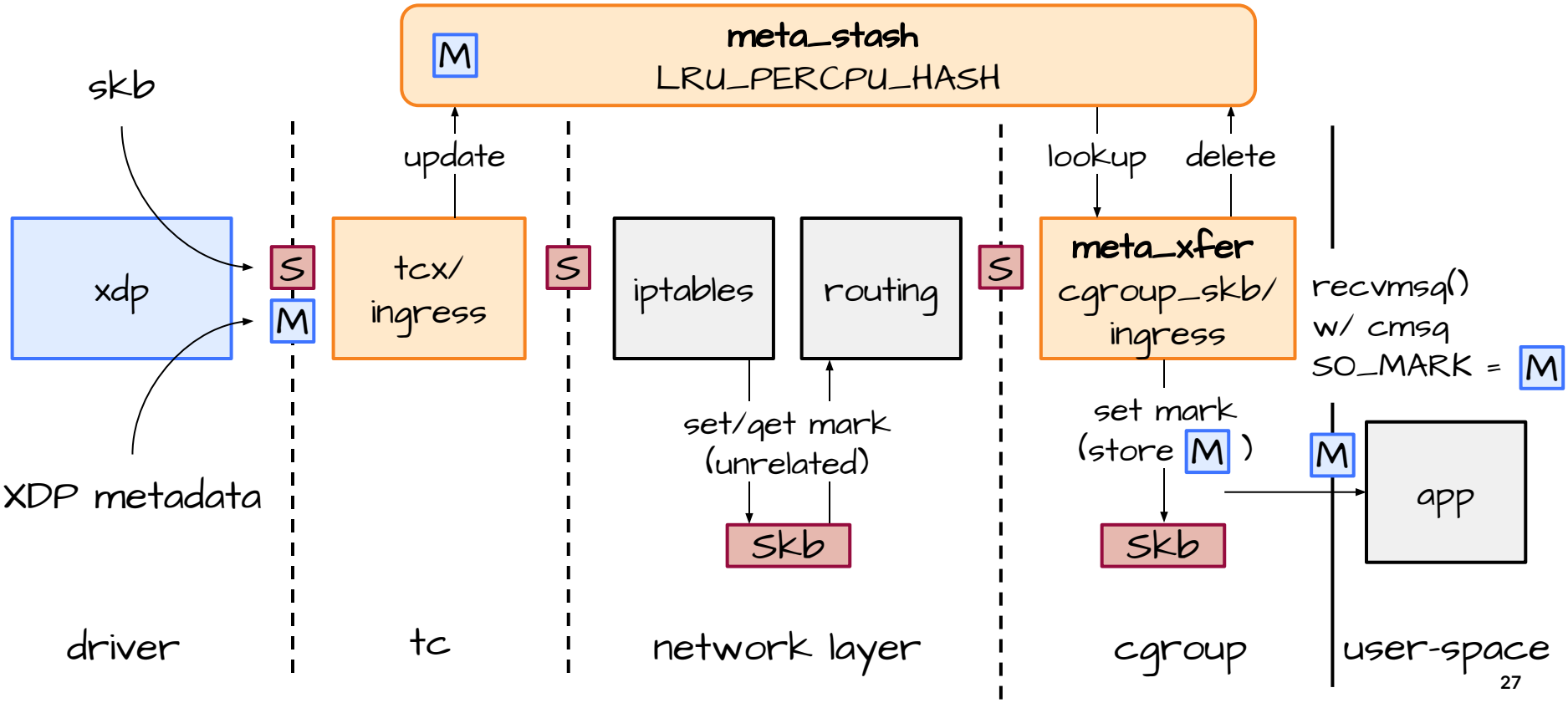
TCP – Baseline: Use BPF map for metadata



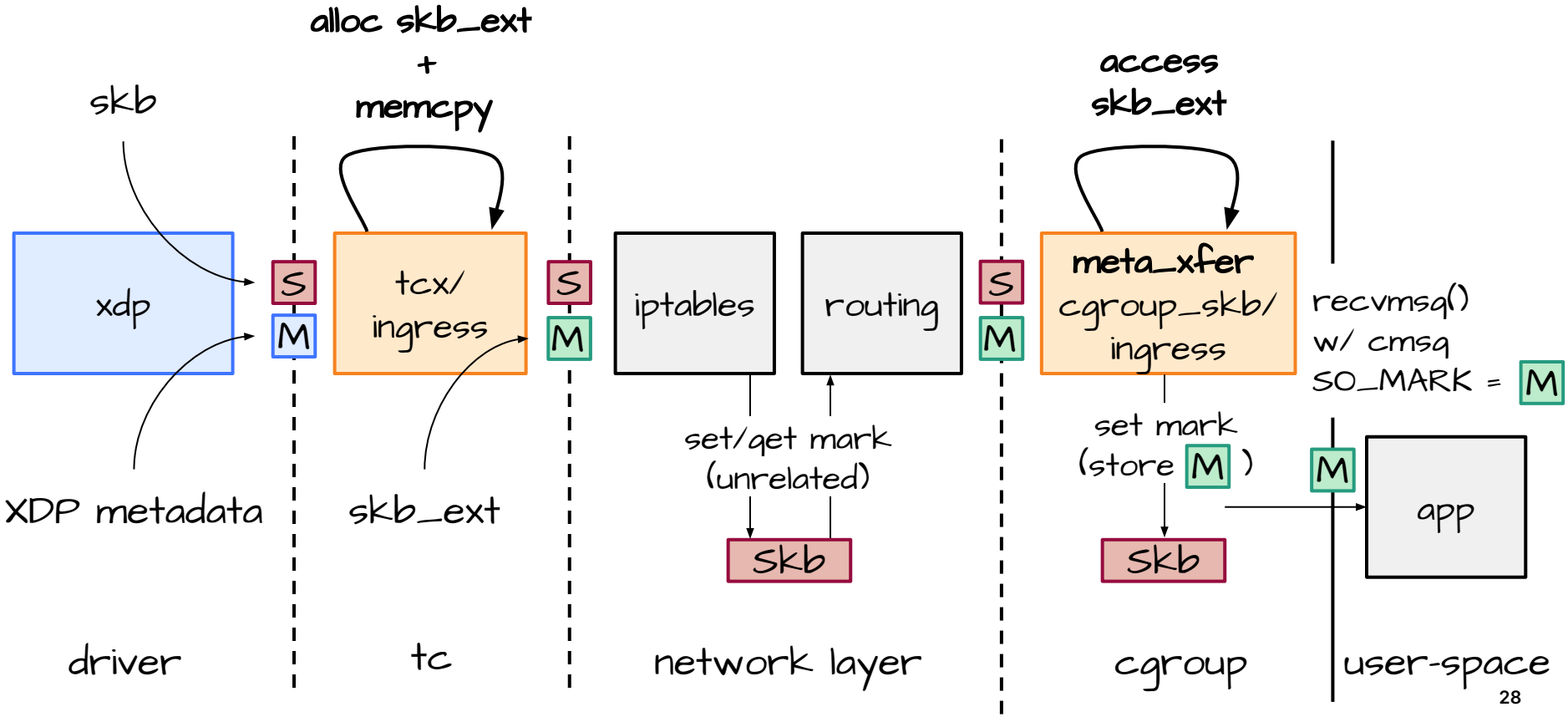
TCP – Experiment: Use BPF skb_ext for metadata



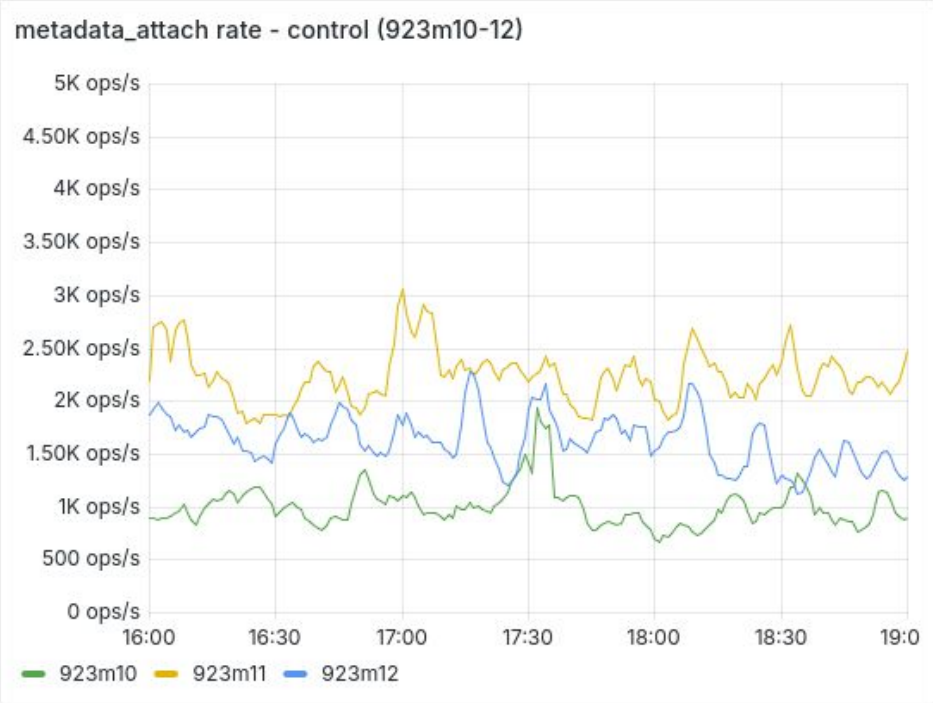
UDP – Baseline: Use BPF map for metadata

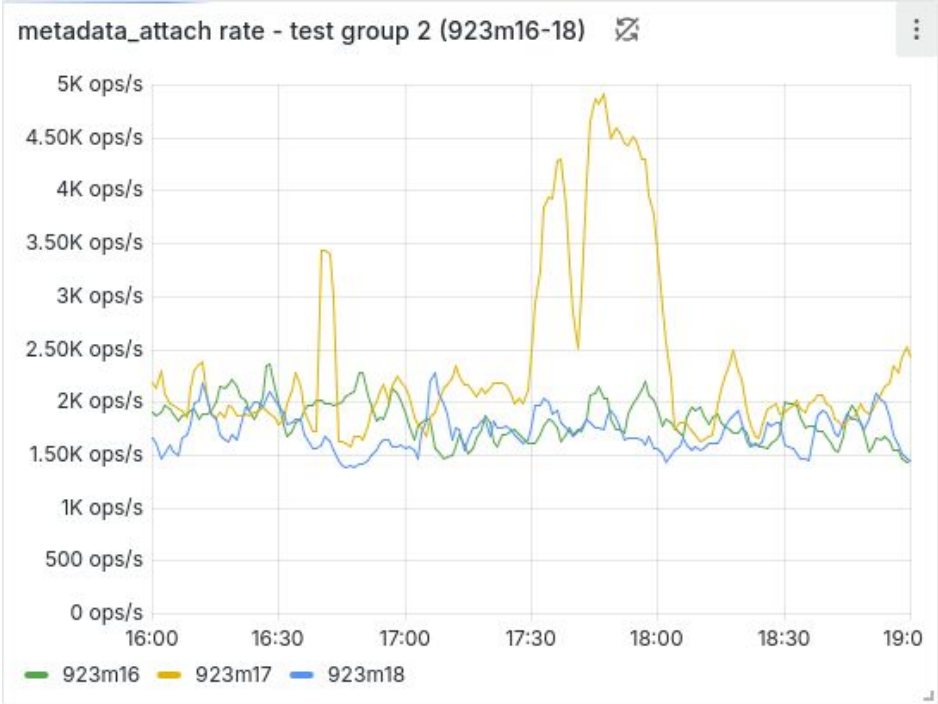


UDP – Experiment: Use BPF skb_ext for metadata

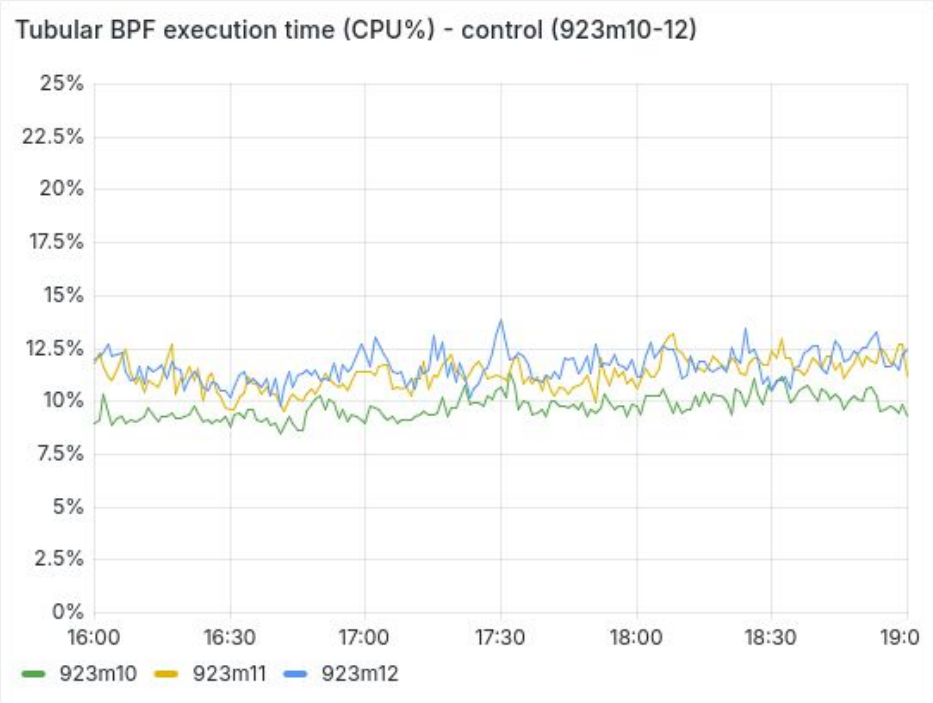


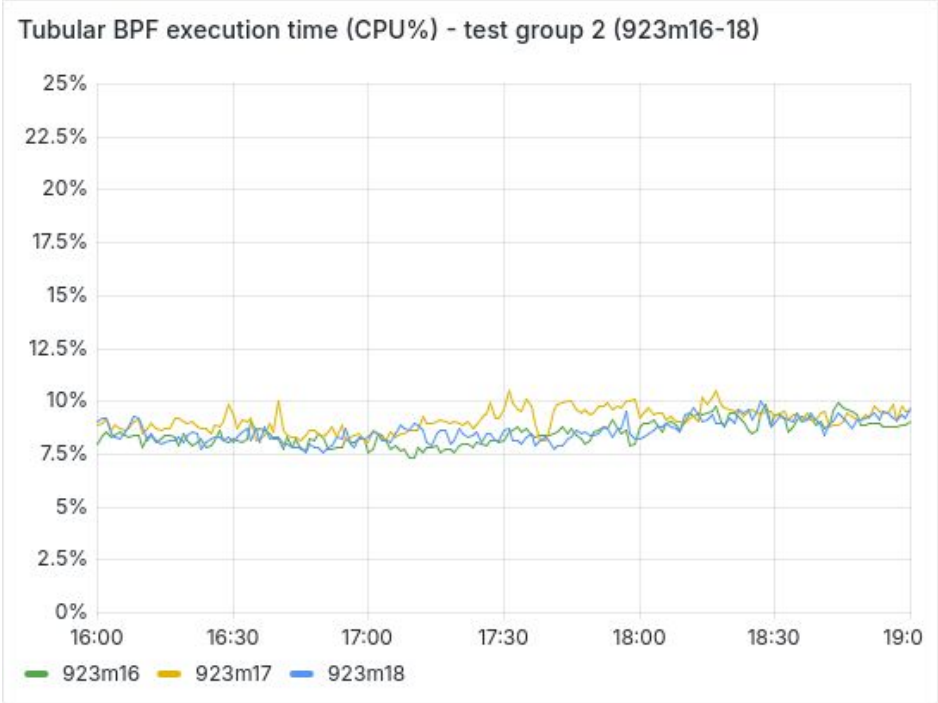
Results: Control vs Test Group 2 (uses skb_ext)
Load comparison





Results: Control vs Test Group 2 (uses skb_ext)
Total runtime for across all involved BPF progs

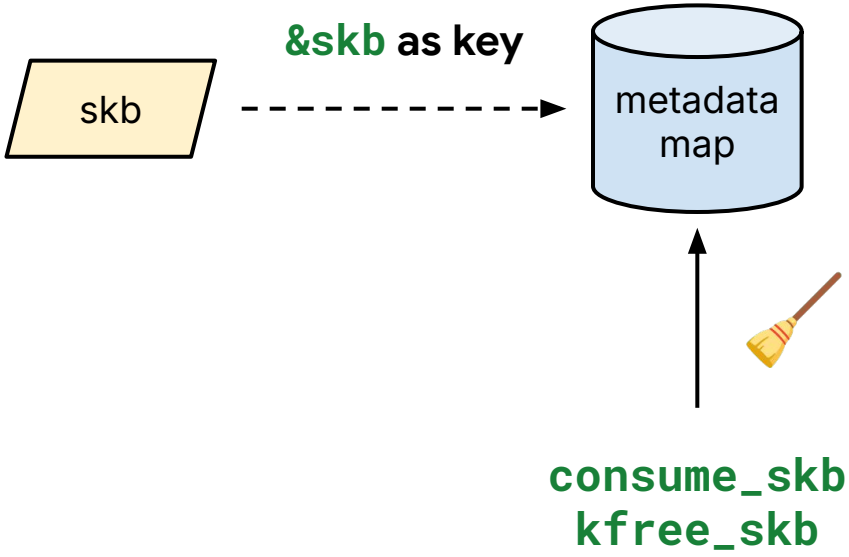




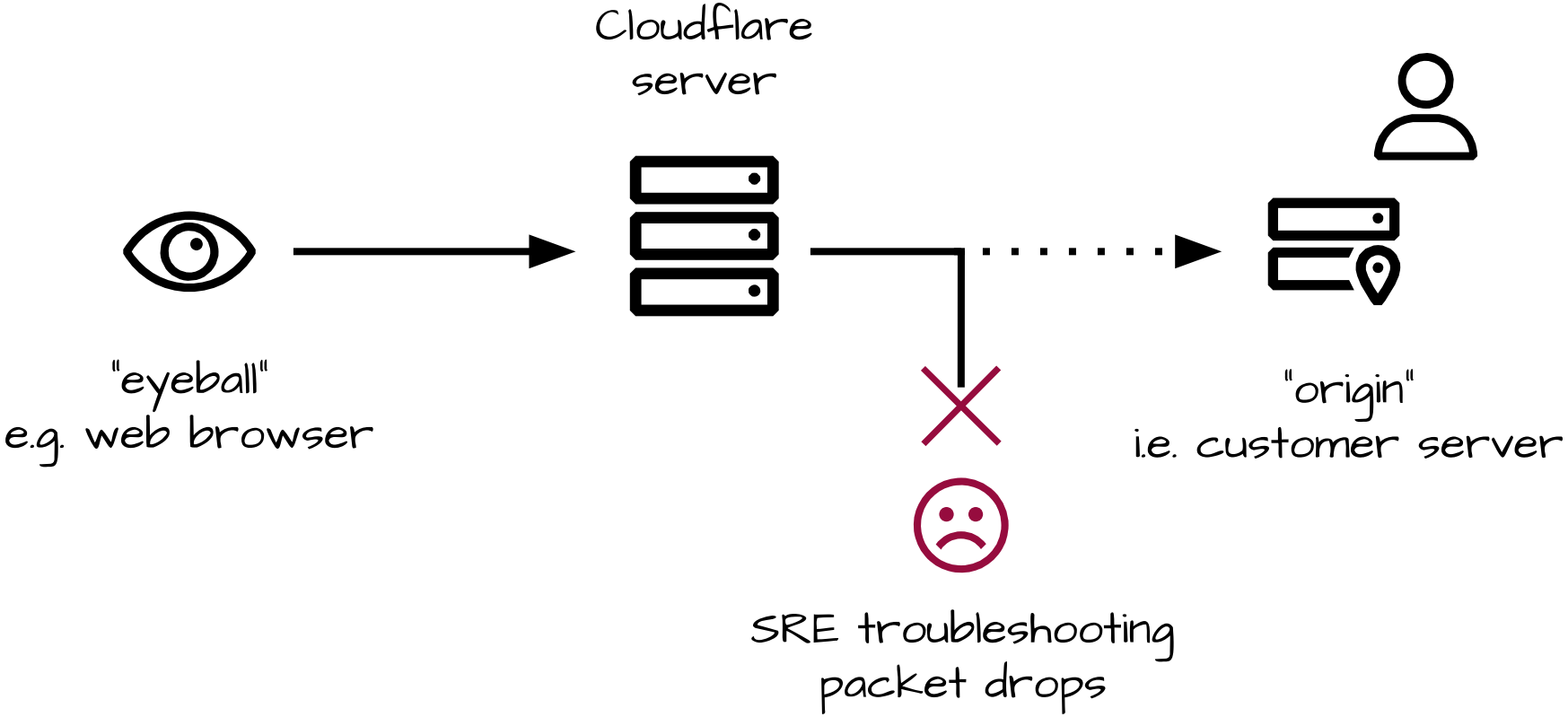
Results: Control vs Test Group 2 (uses skb_ext)
Average runtime per invocation across all involved progs

The alternative approach

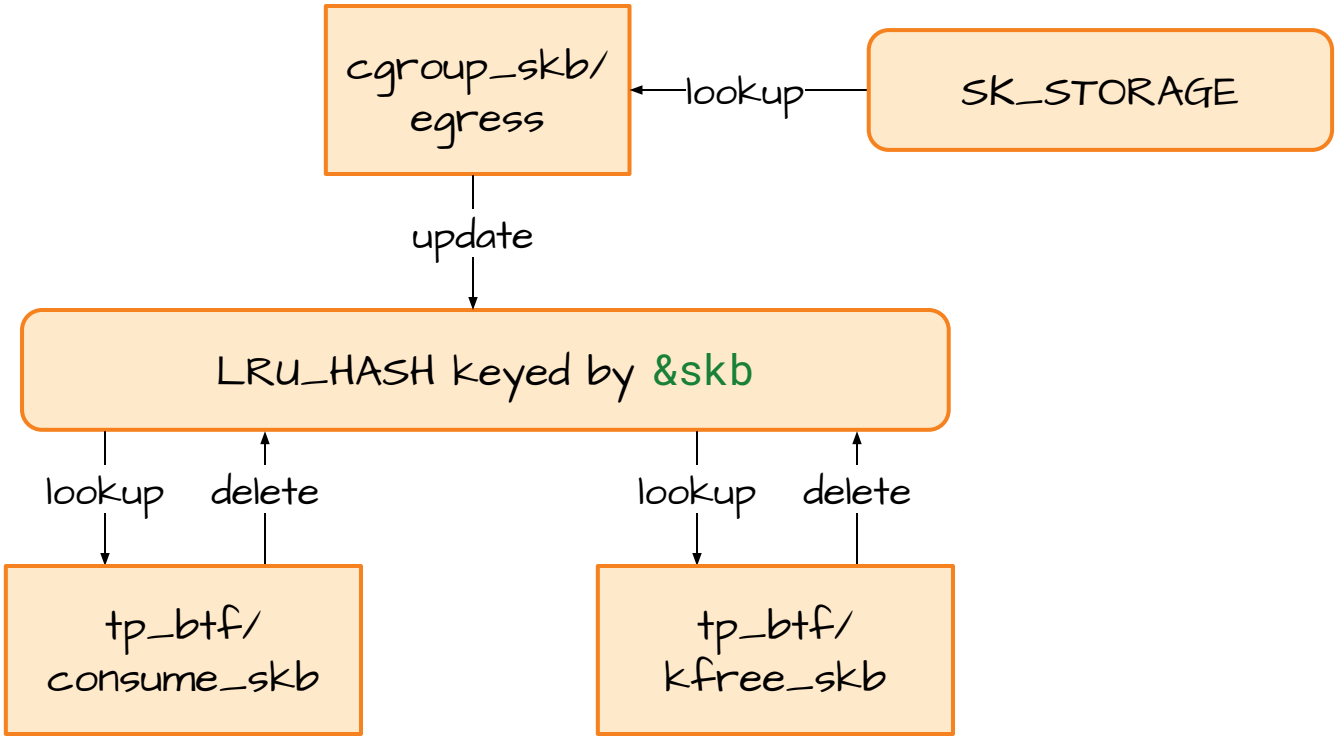
Use BPF map skb side-storage, cleanup from skb destructor tracepoint



Use case – Attribute packet drops to customer ID

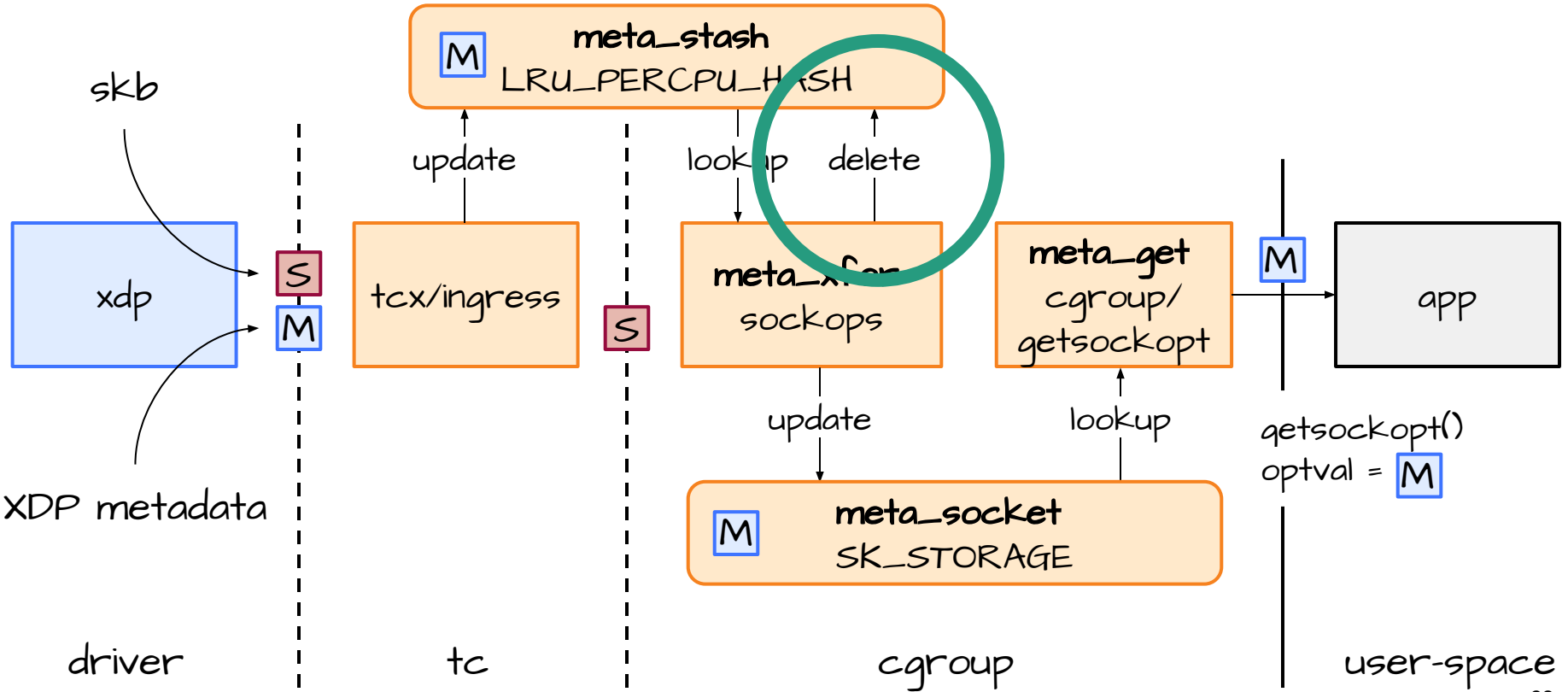


Use case – Attribute packet drops

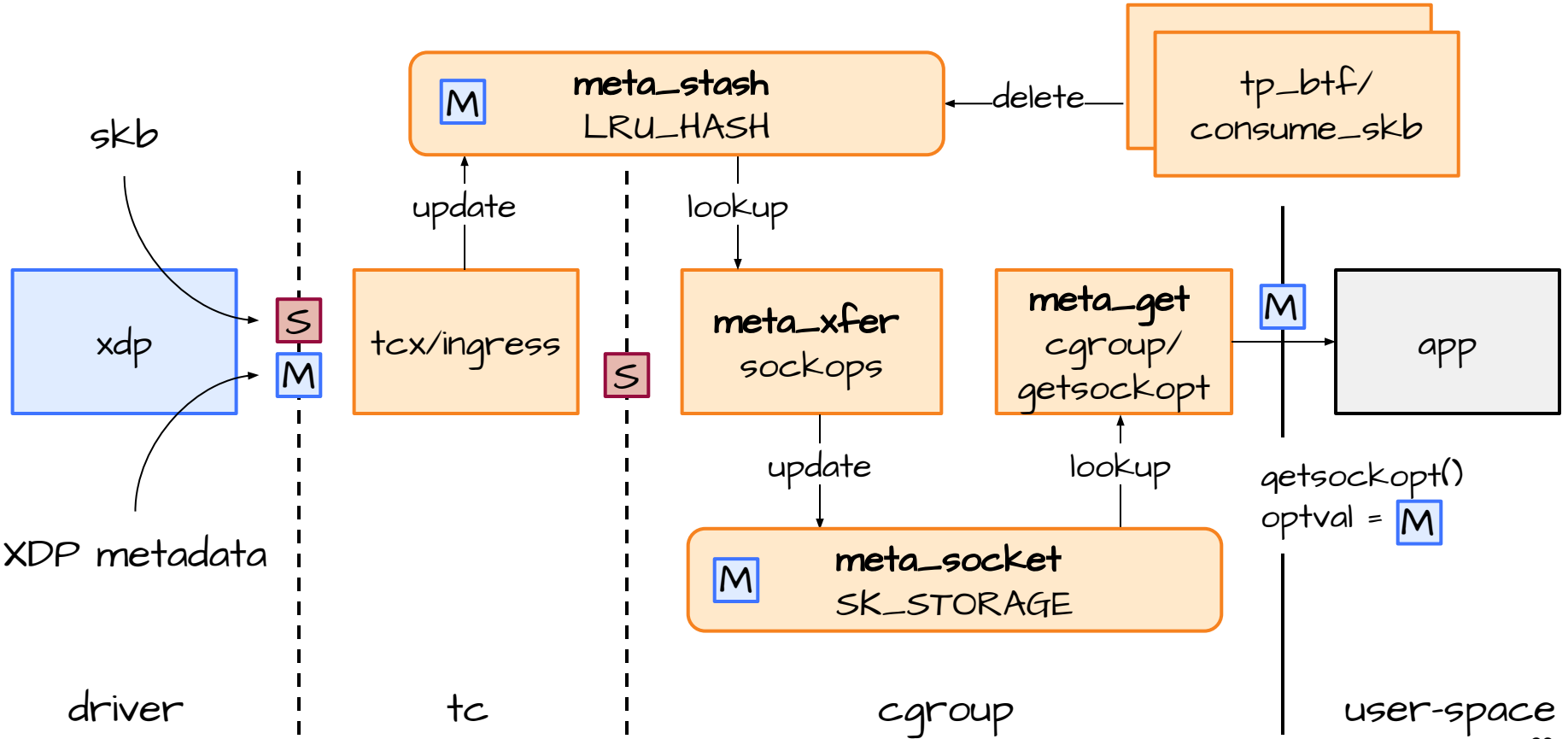


Let’s do another experiment!

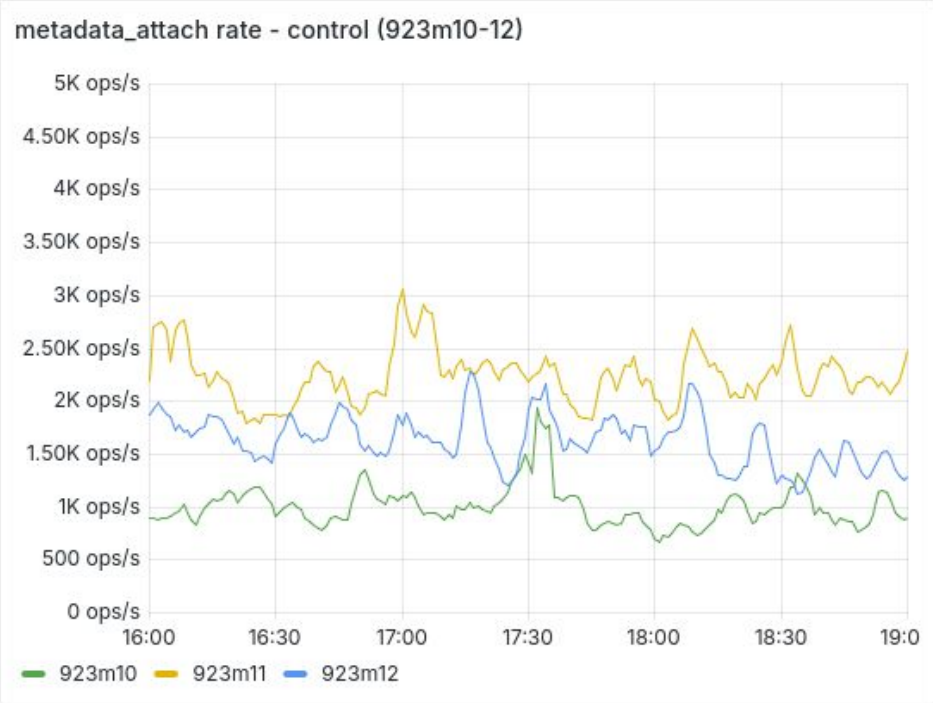
TCP – Baseline: Use BPF map for metadata

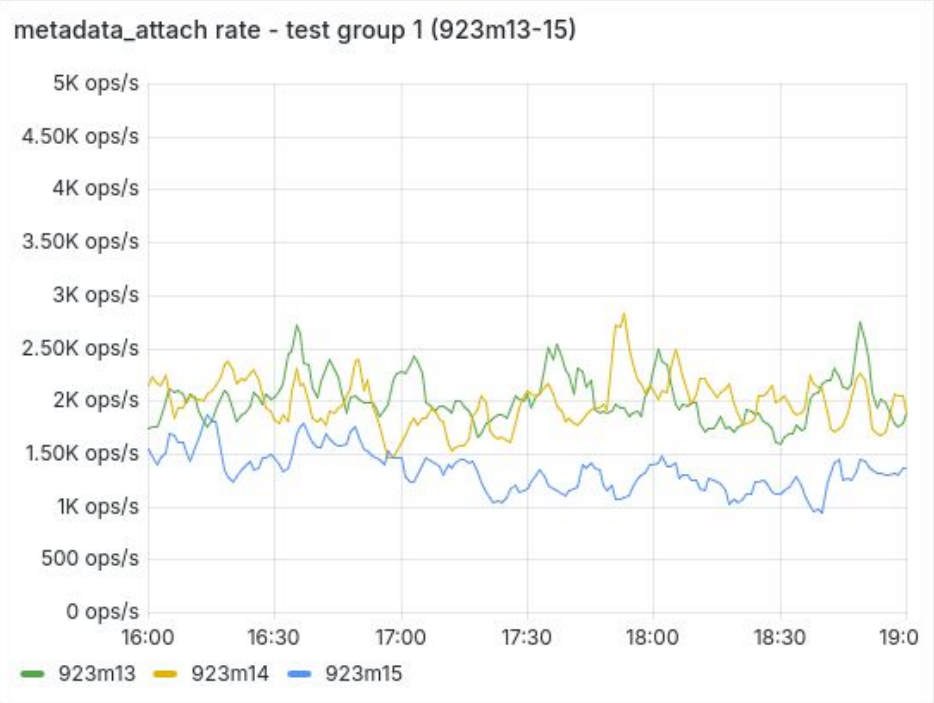


TCP – Alternative: Prune entries from tracepoint

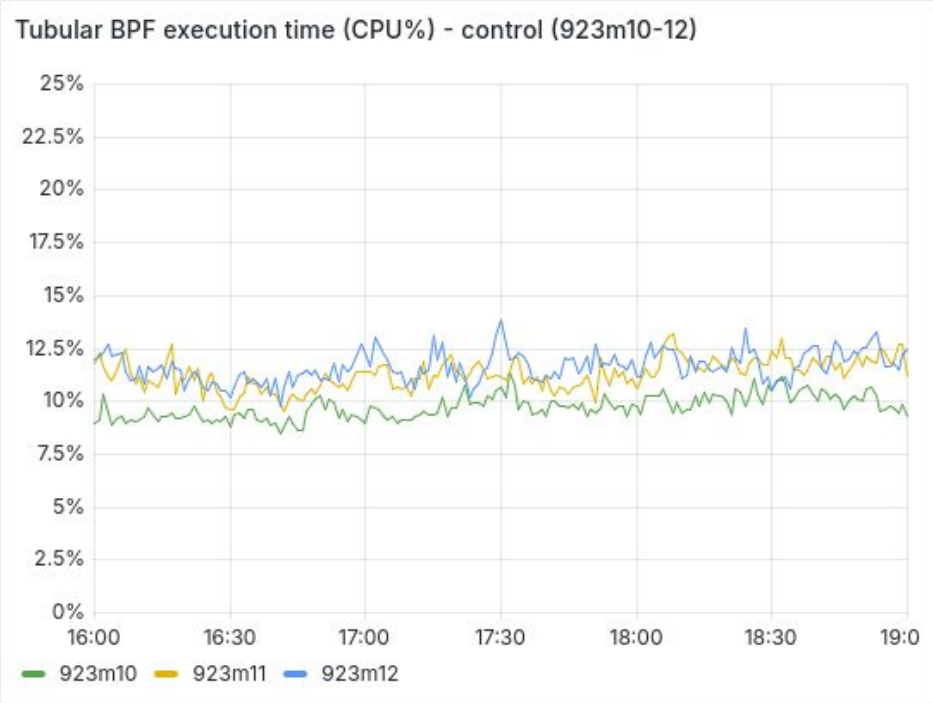


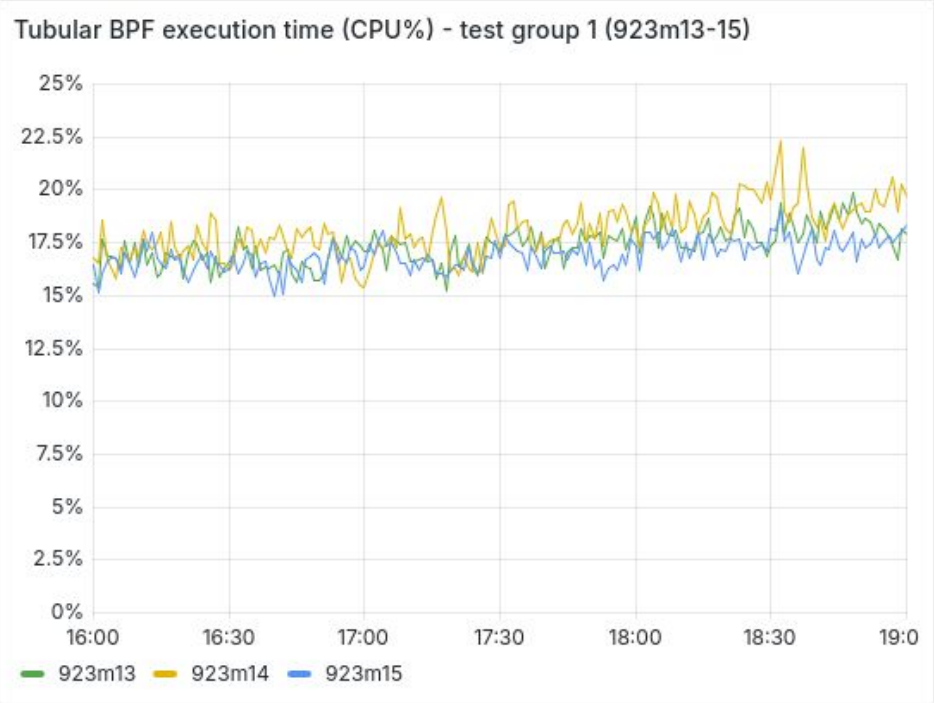
Results: Control vs Test Group 1 (uses tracepoints)
Load comparison



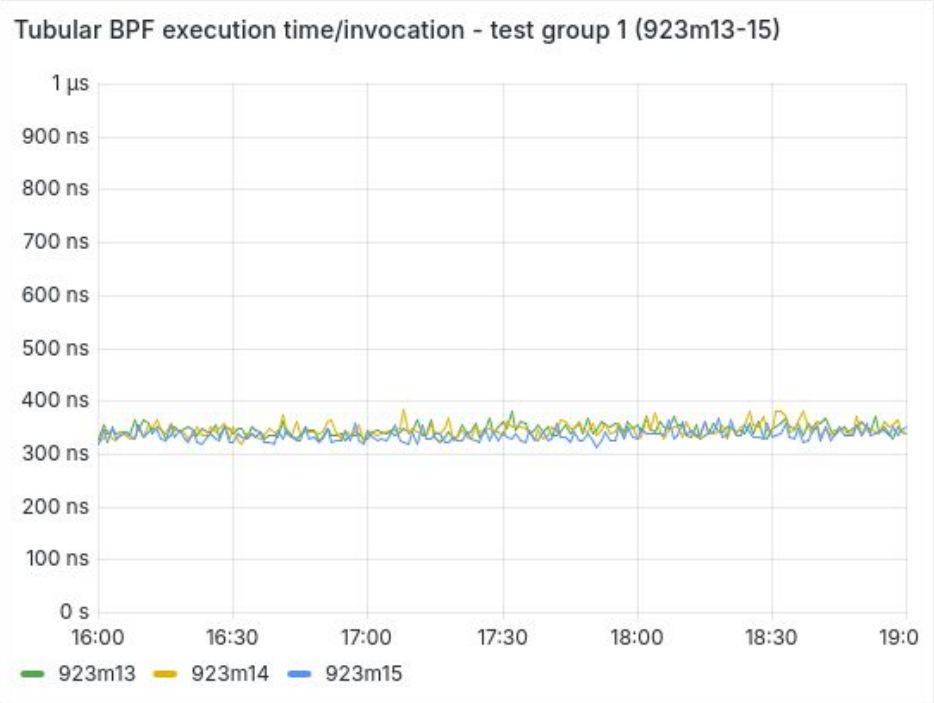


Results: Control vs Test Group 1 (uses tracepoints)
Total runtime for across all involved BPF progs





Results: Control vs Test Group 1 (uses tracepoints)
Average runtime per invocation across all involved progs



WHY SO LOW?

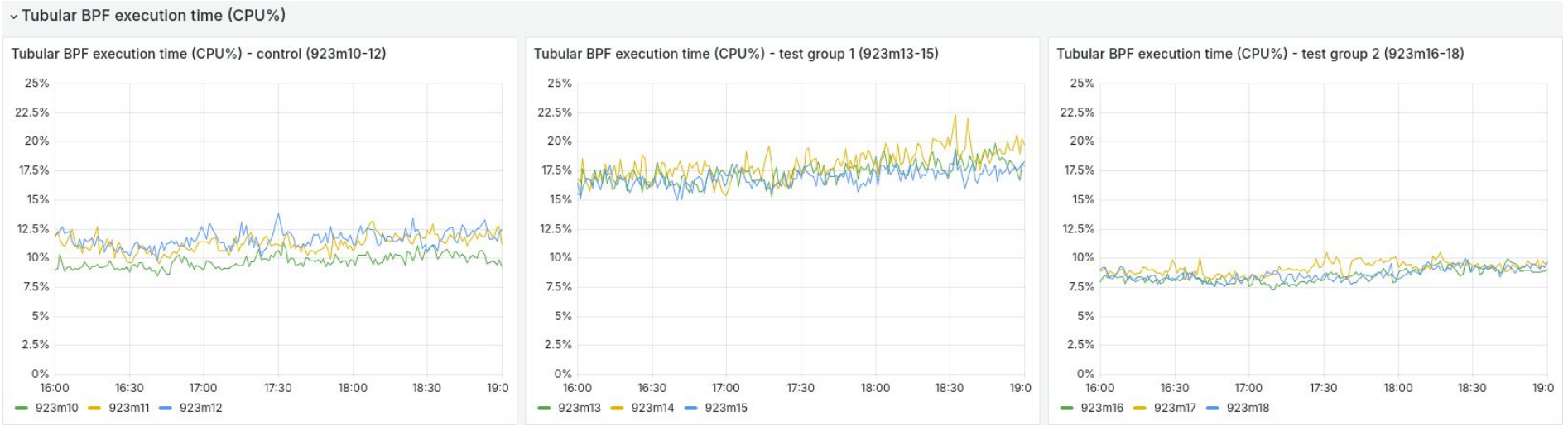
skb consume vs drop vs skb_ext allocations

CLOUDFLARE

```
923m16$ sudo perf stat -a -r 10 \
            -e skb:consume_skb -e skb:kfree_skb \
            -e probe:skb_ext_add -- sleep 1

 Performance counter stats for 'system wide' (10 runs):

           334,256      skb:consume_skb                                 ( +-  1.63% )
             1,525      skb:kfree_skb                                   ( +-  4.09% )
             2,218      probe:skb_ext_add                               ( +- 18.58% )

           1.02305 +- 0.00243 seconds time elapsed  ( +-  0.24% )
```

We attach metadata to <1% of skb's

Side-by-side comparison: Total runtime for across all involved BPF progs



baseline

(metadata cleanup in `skops`, `cgroup_skb/ingress`)

metadata cleanup in `{consume,free}_skb()`

use skb extension for metadata

Open Question: What if we used RHASH?

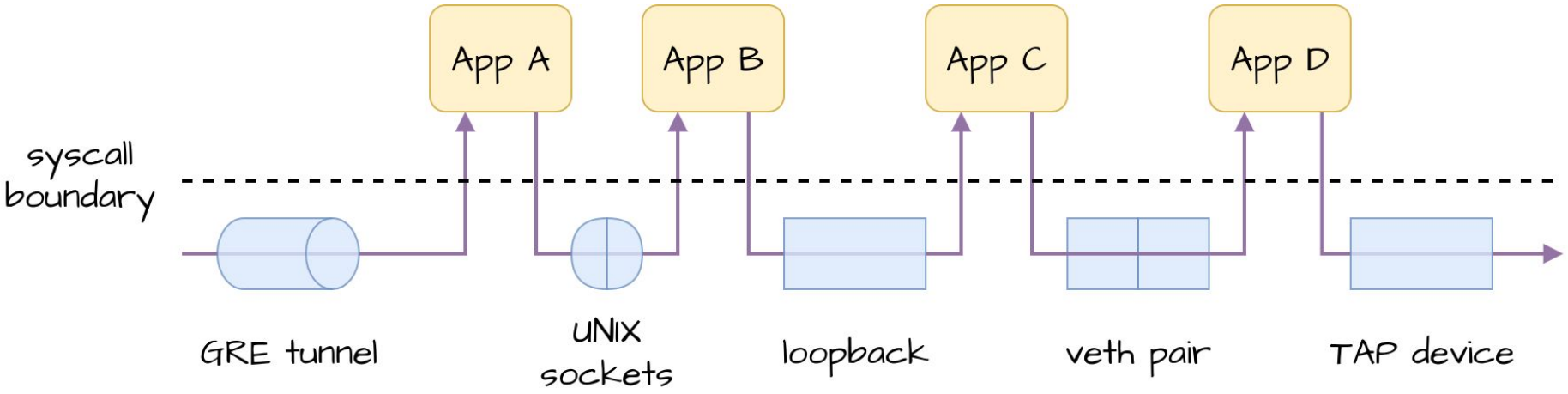
- BPF_MAP_TYPE_RHASH queued for v7.2
- Benchmarks report 5x update speed-up against HASH map
- Is it enough to be same / better than skb_ext-based solution?

https://lore.kernel.org/all/20260605-rhash-v7-0-5b8e05f8630d@meta.com/

CLOUDFLARE

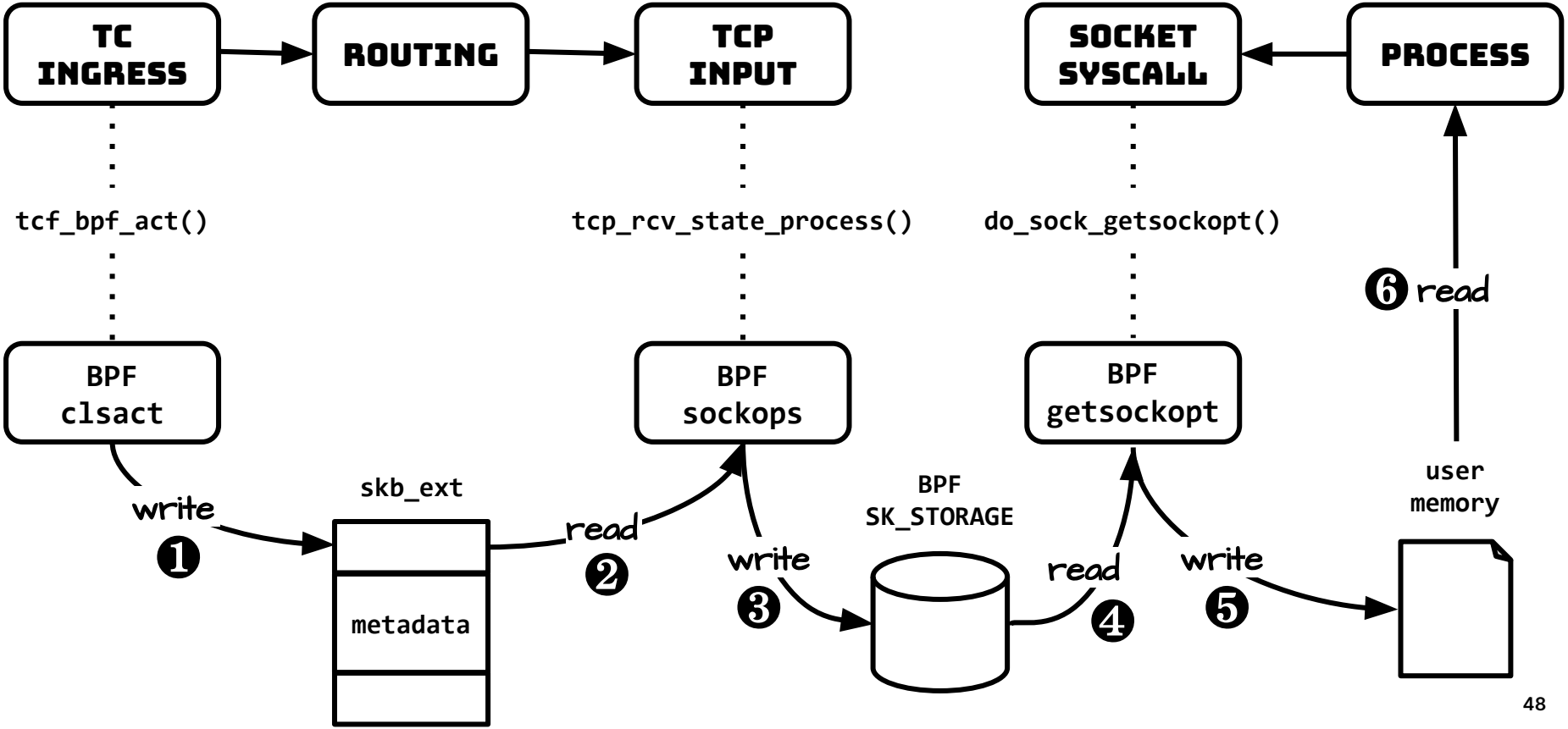
Let’s think about uAPI...

“The Holy Grail” Use Case – Trace L7 requests

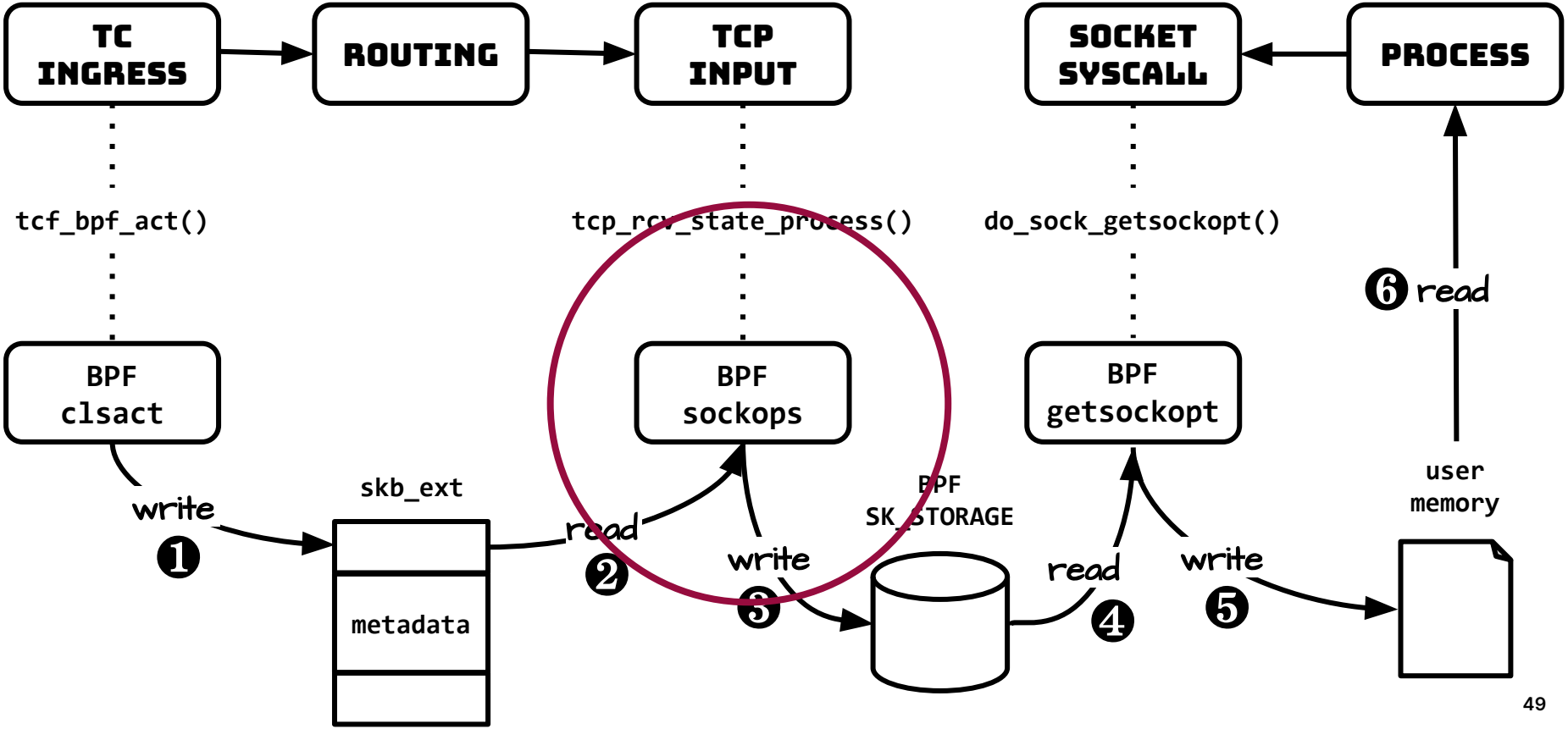


NOT DOABLE WITHOUT COOPERATION WITH USER-SPACE
=> NEED TO BE ABLE TO PASS METADATA IN BOTH DIRECTIONS

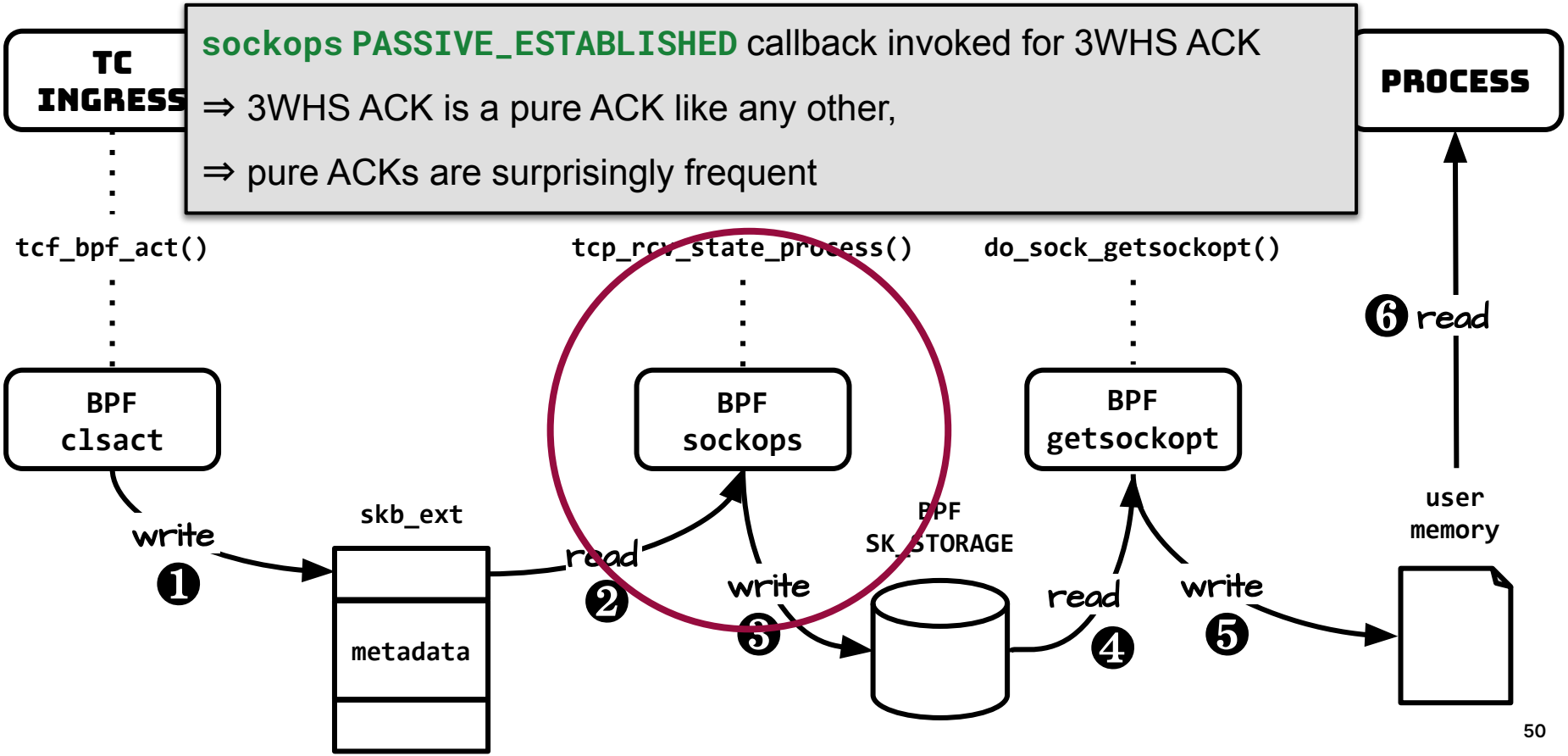
TCP is covered by BPF getsockopt hook



TCP is covered by BPF getsockopt hook



TCP is covered by BPF getsockopt hook



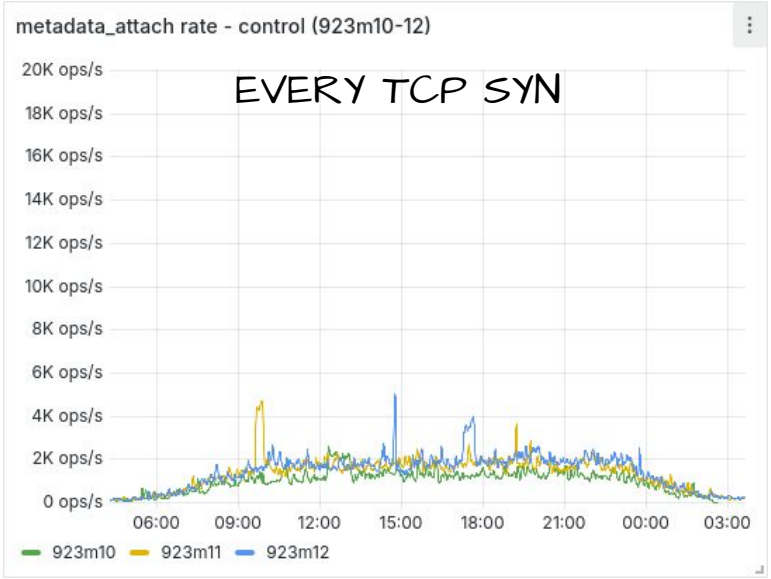
TCP is covered by BPF getsockopt hook

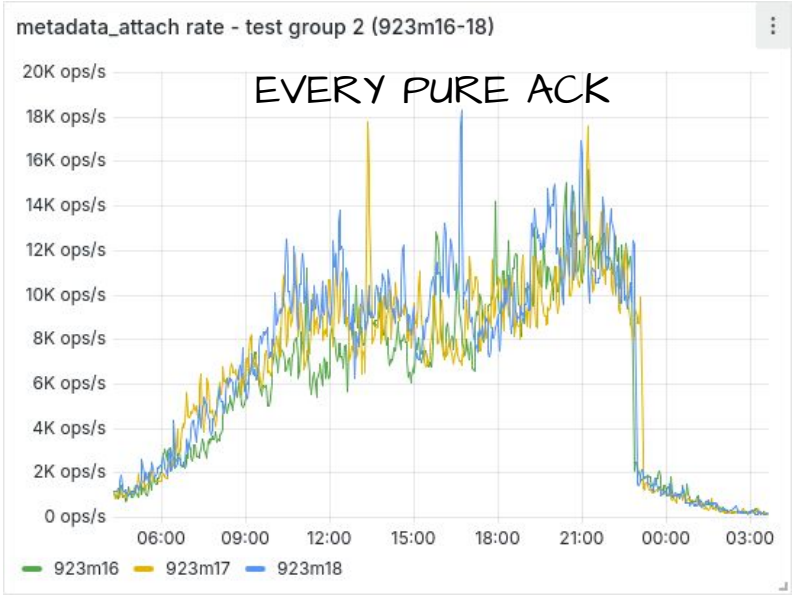
CLOUDFLARE

`sockops` **`PASSIVE_ESTABLISHED`** callback invoked for 3WHS ACK

⇒ 3WHS ACK is a pure ACK like any other,

⇒ pure ACKs are surprisingly frequent





TCP is covered by BPF getsockopt hook

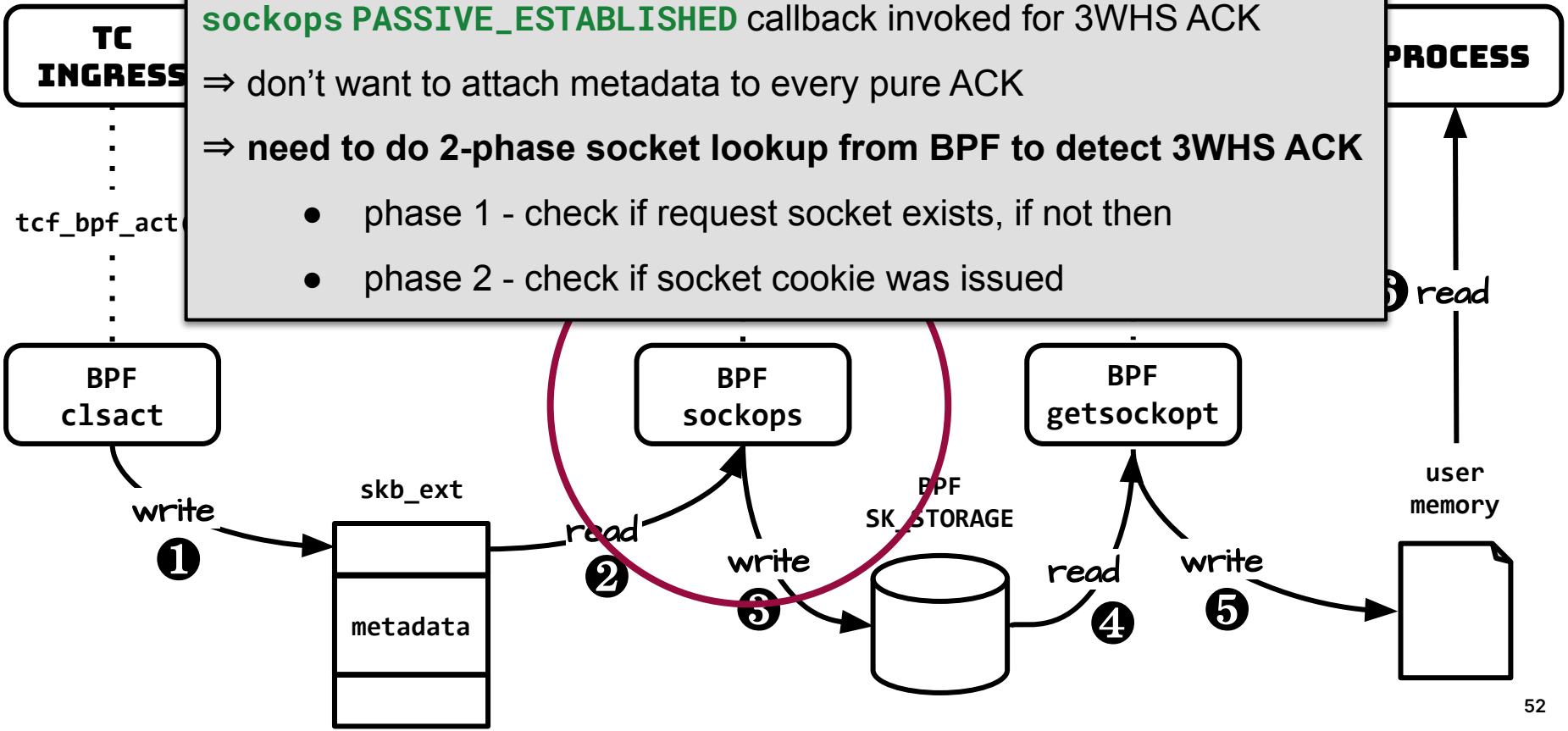
CLOUDFLARE

`sockops PASSIVE_ESTABLISHED` callback invoked for 3WHS ACK

⇒ don't want to attach metadata to every pure ACK

⇒ **need to do 2-phase socket lookup from BPF to detect 3WHS ACK**

- phase 1 - check if request socket exists, if not then
- phase 2 - check if socket cookie was issued



UDP is hard – need to use `cmsg`

CLOUDFLARE

There is no BPF hook to produce a `cmsg`

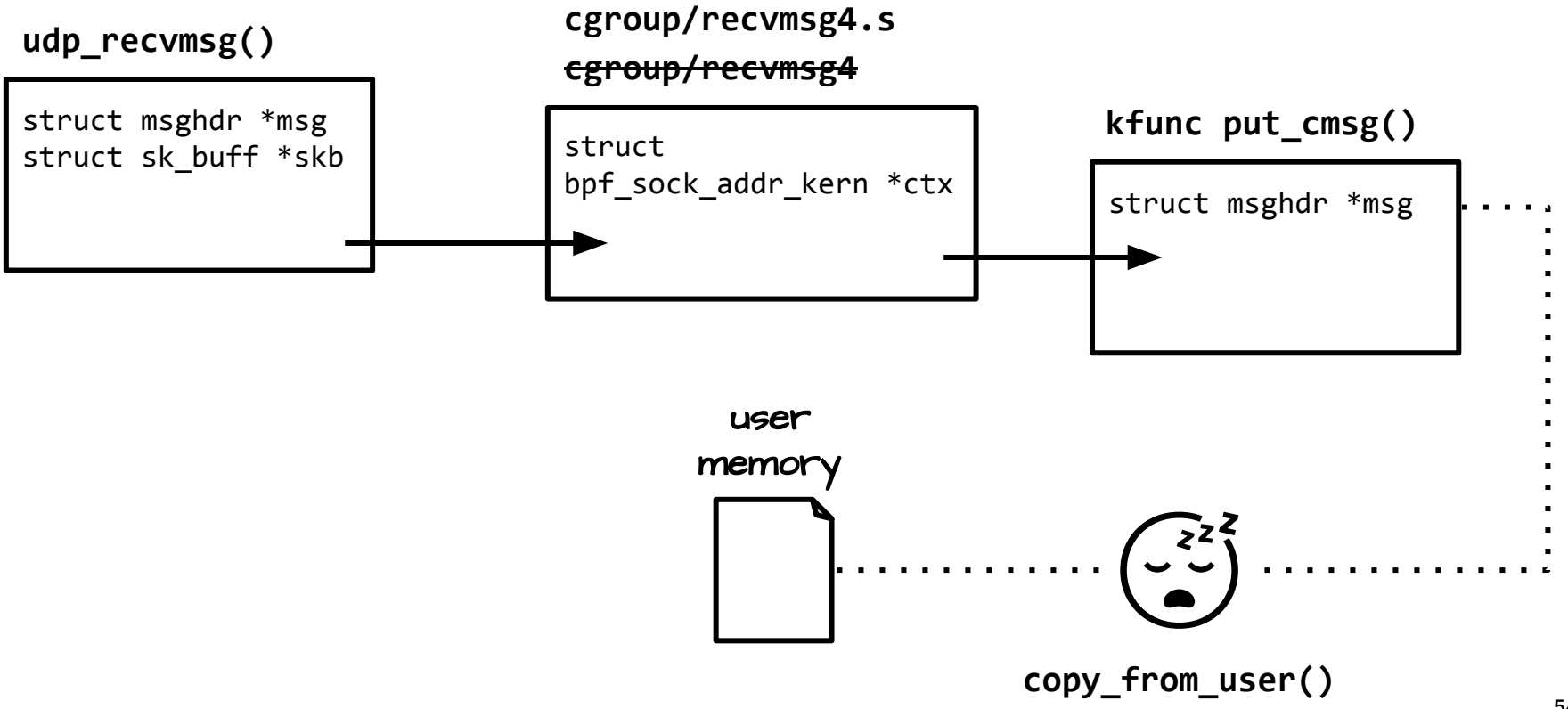
Requires two tracepoint probes:

1. at `fexit:vmlinux:udp_recvmsg`
   poke user memory buffer at `$msg->msg_control_user`
   to write the cmsg contents

... but we also need to update `msg_controllen`
... but `msghdr` is kernel memory at `udp_recvmsg` - can't modify

2. at `fexit:vmlinux:____sys_recvmsg`
   patch `$msg->msg_controllen`

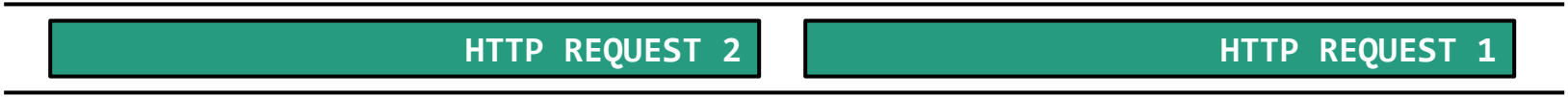
UDP is hard – we have BPF `recvmsg` hook, but...



Even if BPF could pass metadata to user-space, tracing L7 requests is still hard...

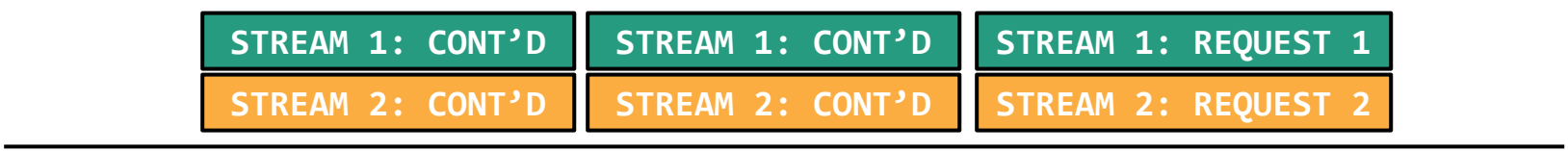
CLOUDFLARE

TCP conn vs HTTP/1.1 connection reuse

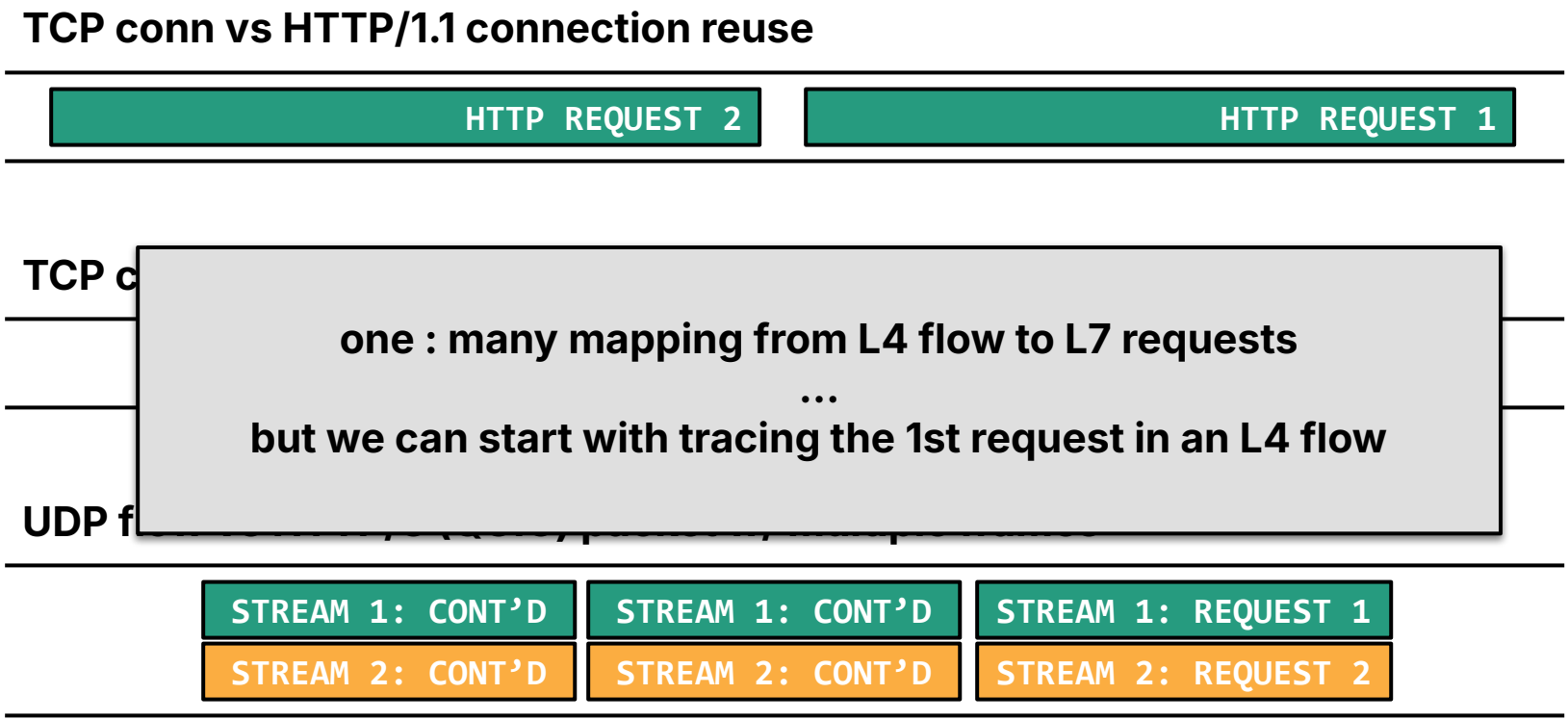


TCP conn vs HTTP/2 stream multiplexing

UDP flow vs HTTP/3 (QUIC) packet w/ multiple frames



Correlating metadata with L7 requests is hard...



💾 Link to RFC patch set

CLOUDFLARE

[PATCH RFC net-next 0/6] skb extension for BPF metadata

https://patch.msgid.link/20260714-bpf-meta-inside-skb-ext-v1-0-5871c07a8dd6@cloudflare.com

Thank you

jakub@cloudflare.com

andrej@cloudflare.com