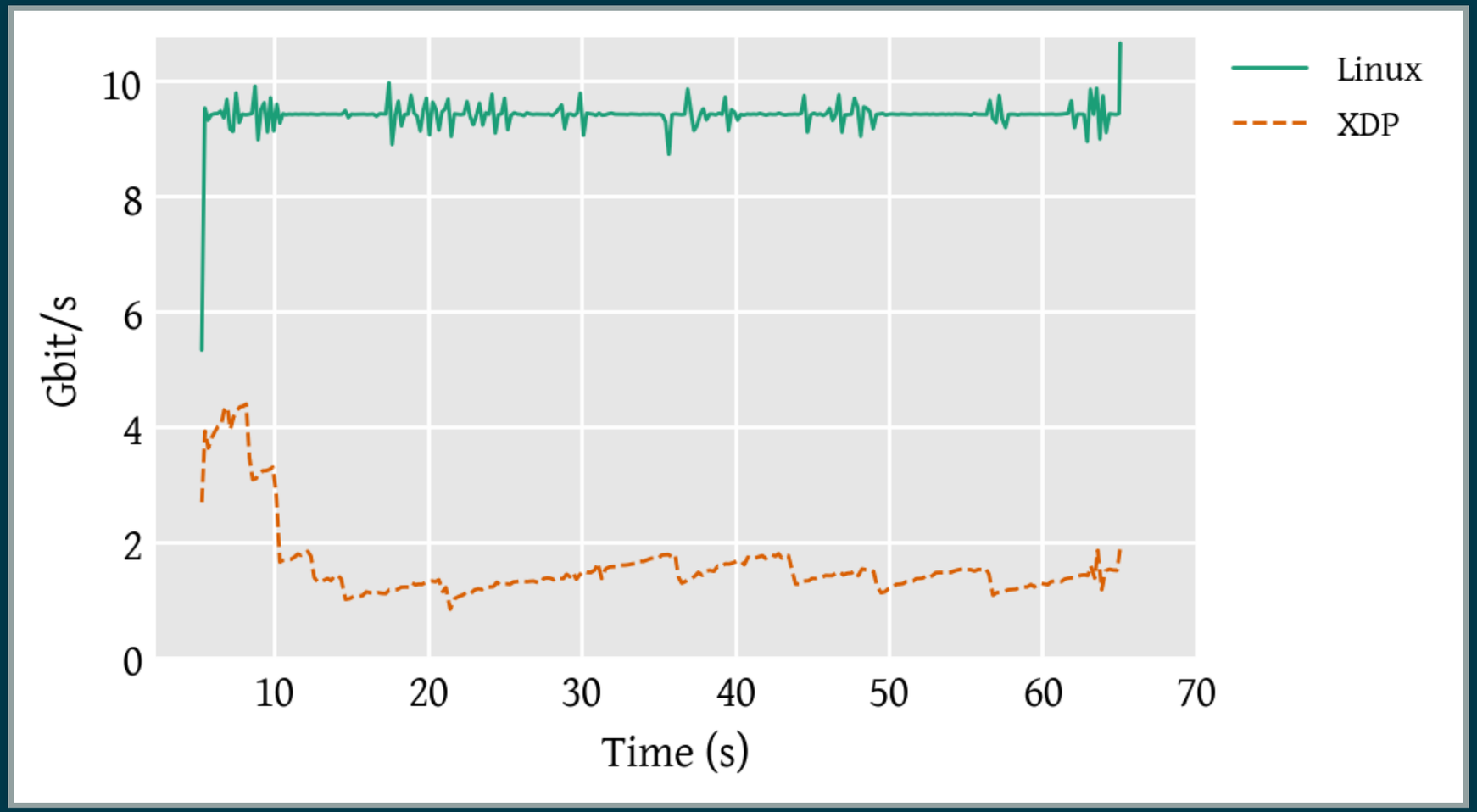
Adding packet queueing to XDP

Toke Høiland-Jørgensen
Principal Kernel Engineer, Red Hat

Linux Plumbers Conference
September 2022


EXCUSE ME, SIR
DO YOU HAVE A FEW MINUTES TO TALK ABOUT QUEUEING?
imgflip.com

Why does XDP need queueing?



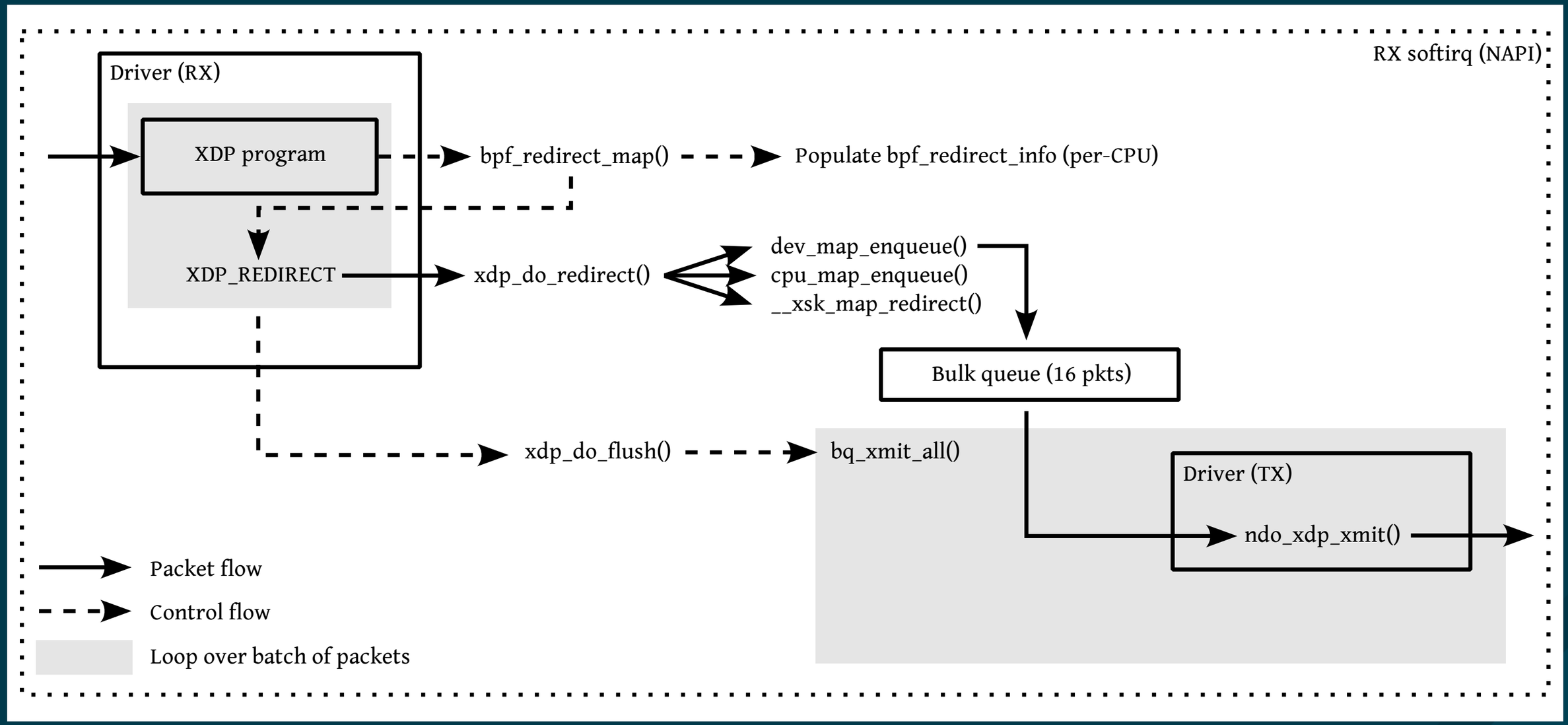
100->10 Gbps rate transition, 10ms base RTT

Why does XDP need queueing? (cont)

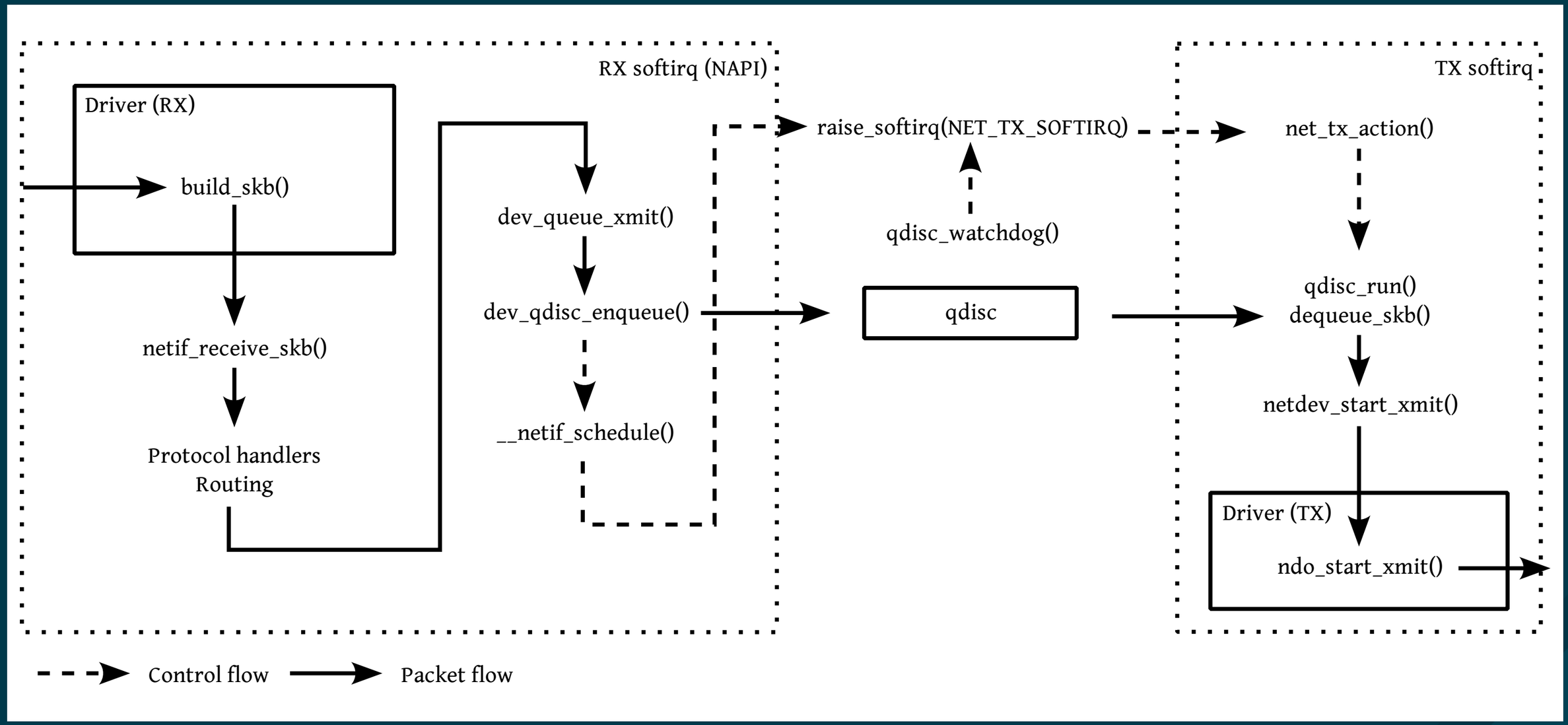
Other use cases enabled by queueing:

- Different packet scheduling policies (FQ, QoS, etc)
- Bandwidth shaping
- Network emulation (reordering, delaying packets)
- Custom buffering schemes (e.g., hold on to packets while spawning container)

Review: How does XDP_REDIRECT work?



Review: Netstack forwarding flow (simplified)



The ingredients we need

- Somewhere to store packets
- A way to schedule dequeue and transmission

Data structures for packet queues

```
$ ls net/sched/sch_*.c | wc -l
38
```

How many different data structures do these 38 qdiscs use?

Data structures for packet queues

```
$ ls net/sched/sch_*.c | wc -l
38
```

How many different data structures do these 38 qdiscs use?

```
struct sk_buff {
        union {
                struct {
                        struct sk_buff          *next;
                        struct sk_buff          *prev;

                        union {
                                struct net_device       *dev;
                                unsigned long           dev_scratch;
                        };
                };
                struct rb_node          rbnode; /* used in netem, ip4 defrag, and tcp stack */
                struct list_head        list;
                struct llist_node       ll_node;
        };
        /* [...] */
};
```

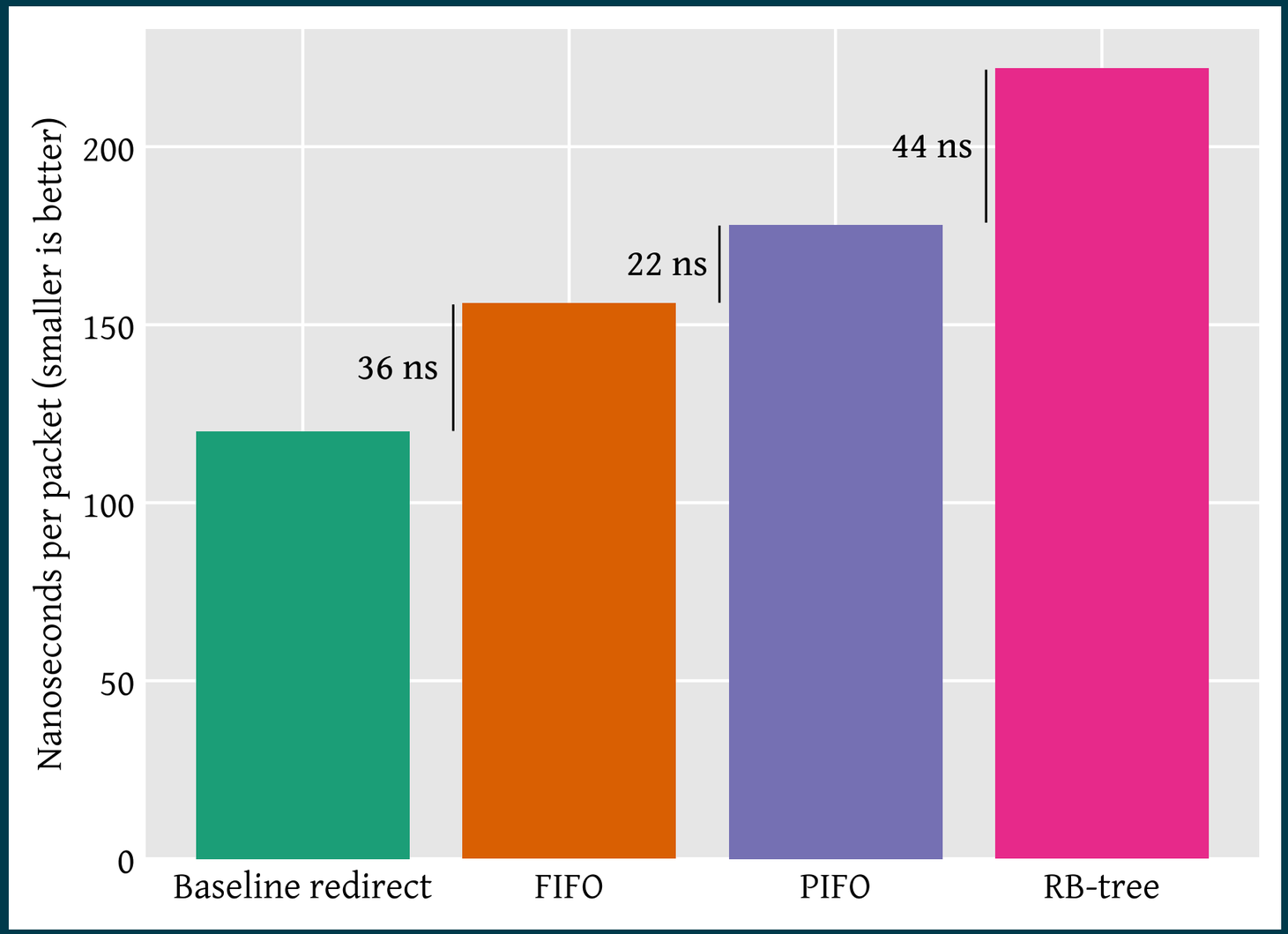
Proposed BPF API

For BPF the natural interface for storing packets is in a map.

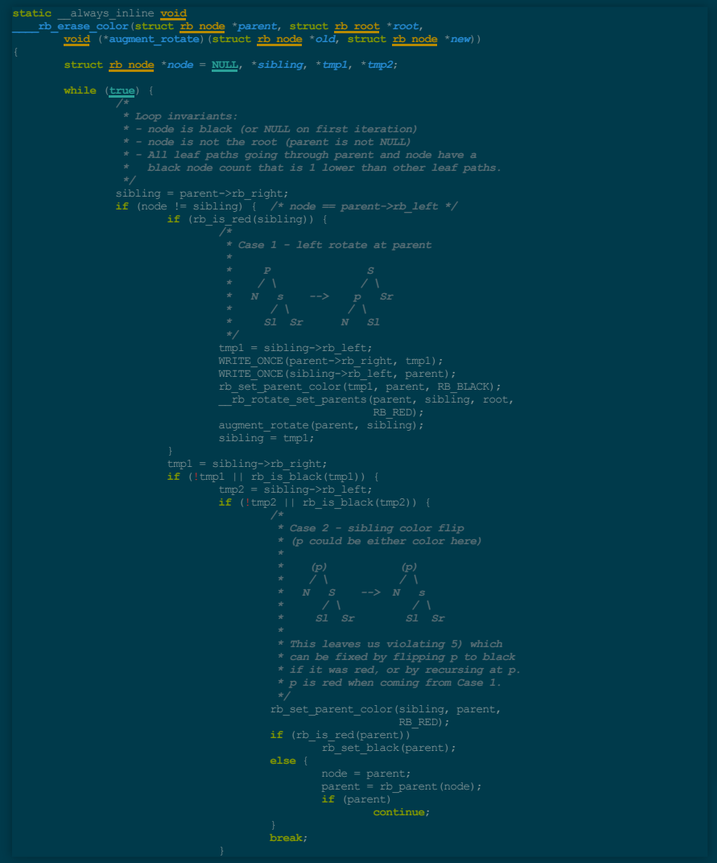
- Implement a priority queue map type for packets (can also be used as FIFO)
- Allow XDP programs to queue packets with `bpf_redirect_map(queue, priority)`
- Create a new `bpf_packet_dequeue()` helper to pull packets out
  - Returns `PTR_TO_BTF_ID` of `struct xdp_md` which can be used like the XDP context

See example code on later slide.

Performance overhead of map types



Where is the RB-tree overhead coming from?



```
static __always_inline void
____rb_erase_color(struct rb_node *parent, struct rb_root *root,
        void (*augment_rotate)(struct rb_node *old, struct rb_node *new))
{
        struct rb_node *node = NULL, *sibling, *tmp1, *tmp2;

        while (true) {
                /*
                 * Loop invariants:
                 * - node is black (or NULL on first iteration)
                 * - node is not the root (parent is not NULL)
                 * - All leaf paths going through parent and node have a
                 *   black node count that is 1 lower than other leaf paths.
                 */
                sibling = parent->rb_right;
                if (node != sibling) {  /* node == parent->rb_left */
                        if (rb_is_red(sibling)) {
                                /*
                                 * Case 1 - left rotate at parent
                                 *
                                 *     P               S
                                 *    / \             / \
                                 *   N   s    -->    P   Sr
                                 *      / \         / \
                                 *     Sl  Sr      N   Sl
                                 */
                                tmp1 = sibling->rb_left;
                                WRITE_ONCE(parent->rb_right, tmp1);
                                WRITE_ONCE(sibling->rb_left, parent);
                                rb_set_parent_color(tmp1, parent, RB_BLACK);
                                __rb_rotate_set_parents(parent, sibling, root,
                                                        RB_RED);
                                augment_rotate(parent, sibling);
                                sibling = tmp1;
                        }
                        tmp1 = sibling->rb_right;
                        if (!tmp1 || rb_is_black(tmp1)) {
                                tmp2 = sibling->rb_left;
                                if (!tmp2 || rb_is_black(tmp2)) {
                                        /*
                                         * Case 2 - sibling color flip
                                         * (p could be either color here)
                                         *
                                         *    (p)           (p)
                                         *    / \           / \
                                         *   N   S    -->  N   s
                                         *      / \           / \
                                         *     Sl  Sr        Sl  Sr
                                         *
                                         * This leaves us violating 5) which
                                         * can be fixed by flipping p to black
                                         * if it was red, or by recursing at p.
                                         * p is red when coming from Case 1.
                                         */
                                        rb_set_parent_color(sibling, parent,
                                                            RB_RED);
                                        if (rb_is_red(parent))
                                                rb_set_black(parent);
                                        else {
                                                node = parent;
                                                parent = rb_parent(node);
                                                if (parent)
                                                        continue;
                                        }
                                        break;
                                }
```

```
                        /*
                         * Case 3 - right rotate at sibling
                         * (p could be either color here)
                         *
                         *   (p)           (p)
                         *   / \           / \
                         *  N   S    -->  N   sl
                         *     / \             \
                         *    sl  Sr            S
                         *                       \
                         *                        Sr
                         *
                         * Note: p might be red, and then both
                         * p and sl are red after rotation(which
                         * breaks property 4). This is fixed in
                         * Case 4 (in __rb_rotate_set_parents()
                         *         which set sl the color of p
                         *         and set p RB_BLACK)
                         *
                         *   (p)           (sl)
                         *   / \           /  \
                         *  N   sl  -->   P    S
                         *       \       /      \
                         *        S     N        Sr
                         *         \
                         *          Sr
                         */
                        tmp1 = tmp2->rb_right;
                        WRITE_ONCE(sibling->rb_left, tmp1);
                        WRITE_ONCE(tmp2->rb_right, sibling);
                        WRITE_ONCE(parent->rb_right, tmp2);
                        if (tmp1)
                                rb_set_parent_color(tmp1, sibling,
                                                    RB_BLACK);
                        augment_rotate(sibling, tmp2);
                        tmp1 = sibling;
                        sibling = tmp2;
                }
                /*
                 * Case 4 - left rotate at parent + color flips
                 * (p and sl could be either color here.
                 *  After rotation, p becomes black, s acquires
                 *  p's color, and sl keeps its color)
                 *
                 *      (p)           (s)
                 *      / \           / \
                 *     N   S     -->  P   Sr
                 *        / \       / \
                 *      (sl) sr     N  (sl)
                 */
                tmp2 = sibling->rb_left;
                WRITE_ONCE(parent->rb_right, tmp2);
                WRITE_ONCE(sibling->rb_left, parent);
                rb_set_parent_color(tmp1, sibling, RB_BLACK);
                if (tmp2)
                        rb_set_parent(tmp2, parent);
                __rb_rotate_set_parents(parent, sibling, root,
                                        RB_BLACK);
                augment_rotate(parent, sibling);
                break;
```

```
        } else {
                sibling = parent->rb_left;
                if (rb_is_red(sibling)) {
                        /* Case 1 - right rotate at parent */
                        tmp1 = sibling->rb_right;
                        WRITE_ONCE(parent->rb_left, tmp1);
                        WRITE_ONCE(sibling->rb_right, parent);
                        rb_set_parent_color(tmp1, parent, RB_BLACK);
                        __rb_rotate_set_parents(parent, sibling, root,
                                                RB_RED);
                        augment_rotate(parent, sibling);
                        sibling = tmp1;
                }
                tmp1 = sibling->rb_left;
                if (!tmp1 || rb_is_black(tmp1)) {
                        tmp2 = sibling->rb_right;
                        if (!tmp2 || rb_is_black(tmp2)) {
                                /* Case 2 - sibling color flip */
                                rb_set_parent_color(sibling, parent,
                                                    RB_RED);
                                if (rb_is_red(parent))
                                        rb_set_black(parent);
                                else {
                                        node = parent;
                                        parent = rb_parent(node);
                                        if (parent)
                                                continue;
                                }
                                break;
                        }
                        /* Case 3 - left rotate at sibling */
                        tmp1 = tmp2->rb_left;
                        WRITE_ONCE(sibling->rb_right, tmp1);
                        WRITE_ONCE(tmp2->rb_left, sibling);
                        WRITE_ONCE(parent->rb_left, tmp2);
                        if (tmp1)
                                rb_set_parent_color(tmp1, sibling,
                                                    RB_BLACK);
                        augment_rotate(sibling, tmp2);
                        tmp1 = sibling;
                        sibling = tmp2;
                }
                /* Case 4 - right rotate at parent + color flips */
                tmp2 = sibling->rb_right;
                WRITE_ONCE(parent->rb_left, tmp2);
                WRITE_ONCE(sibling->rb_right, parent);
                rb_set_parent_color(tmp1, sibling, RB_BLACK);
                if (tmp2)
                        rb_set_parent(tmp2, parent);
                __rb_rotate_set_parents(parent, sibling, root,
                                        RB_BLACK);
                augment_rotate(parent, sibling);
                break;
        }
    }
}
```

Sidetrack: PIFO queues

In the literature, the Push-In, First-Out (PIFO) queue appeared in 2016 [0].

- It's a limited priority queue (only dequeue at head)
  - Can be implemented in silicon
- We don't need to limit ourselves to the PIFO, however:
  - We can use an optimised algorithm for software by Saeed et al [1]

[0] Sivaraman et al, 2016: "Programmable Packet Scheduling at Line Rate"

[1] Saeed et al, 2019: "Eiffel: Eifficient and Flexible Packet Scheduling"

The Eiffel PIFO algorithm

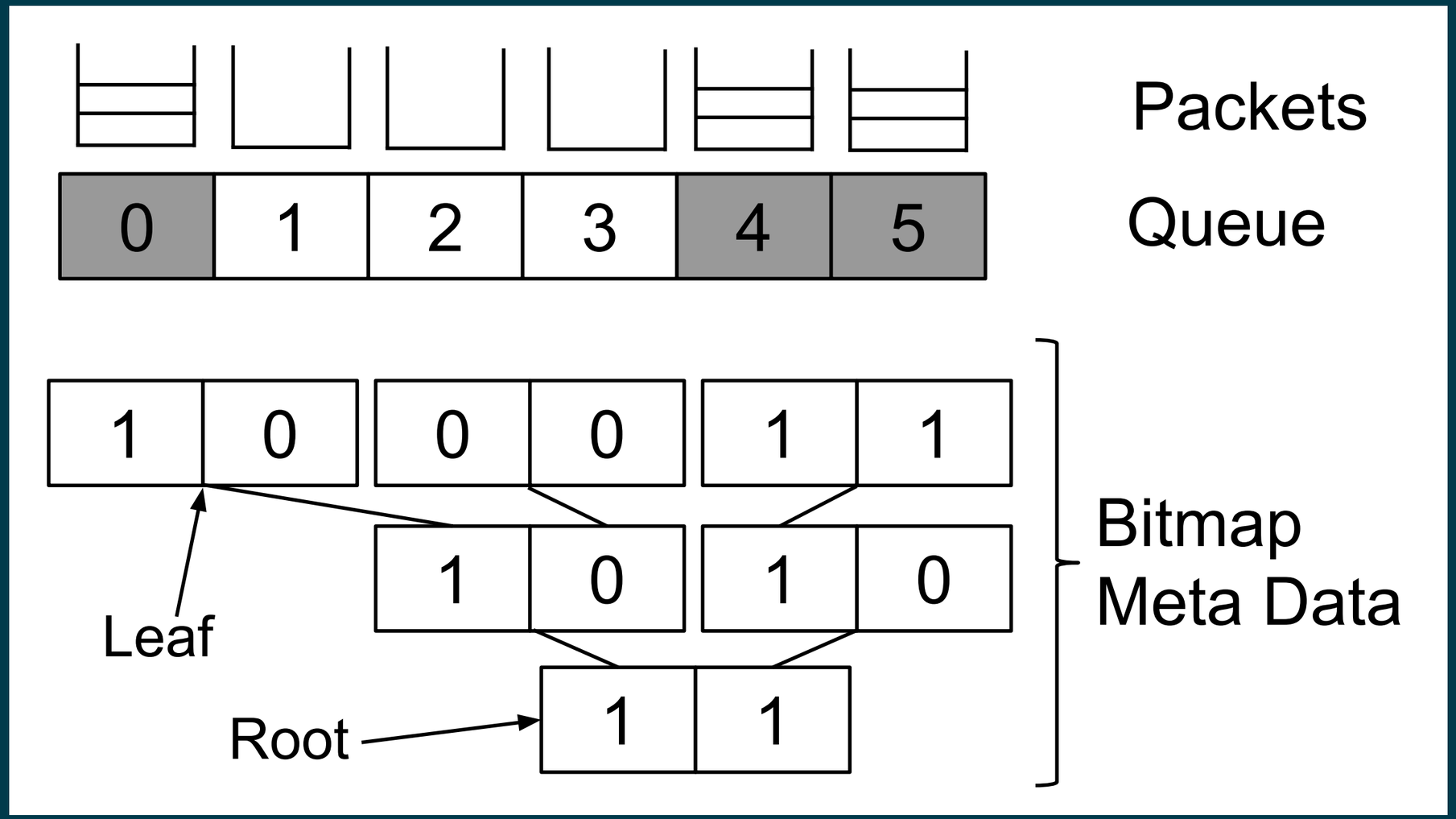


Figure 3 from Saeed et al (2019)

The Eiffel PIFO: rotating queues

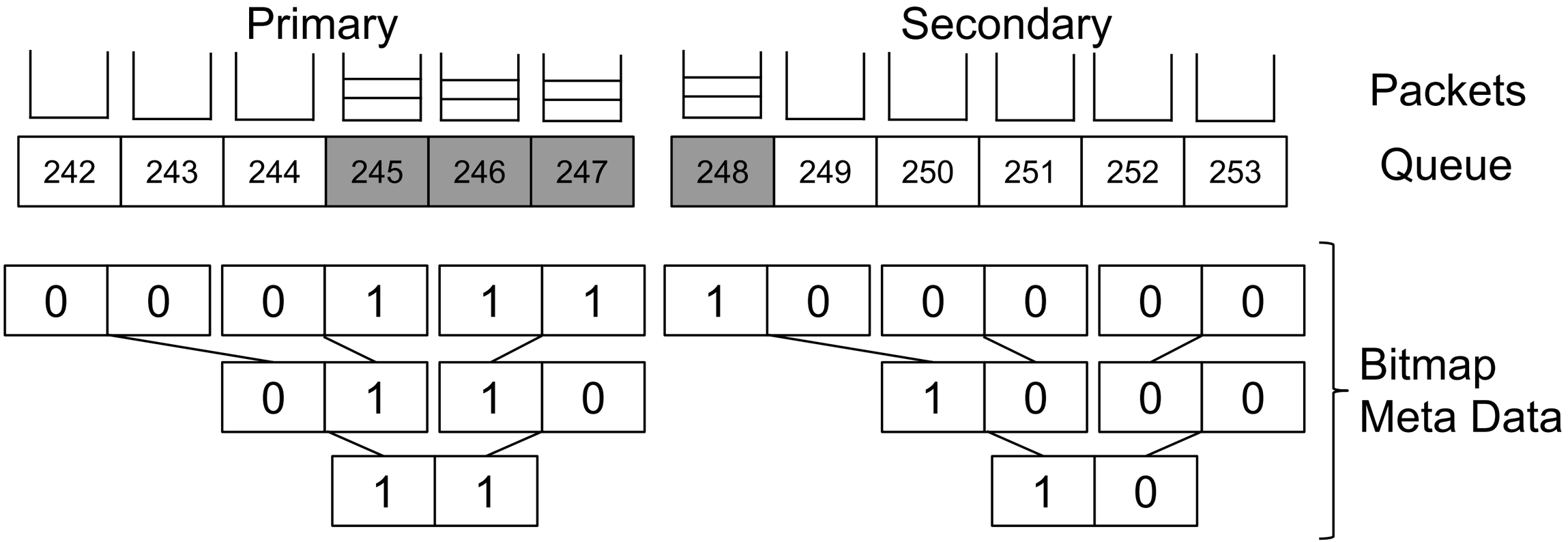
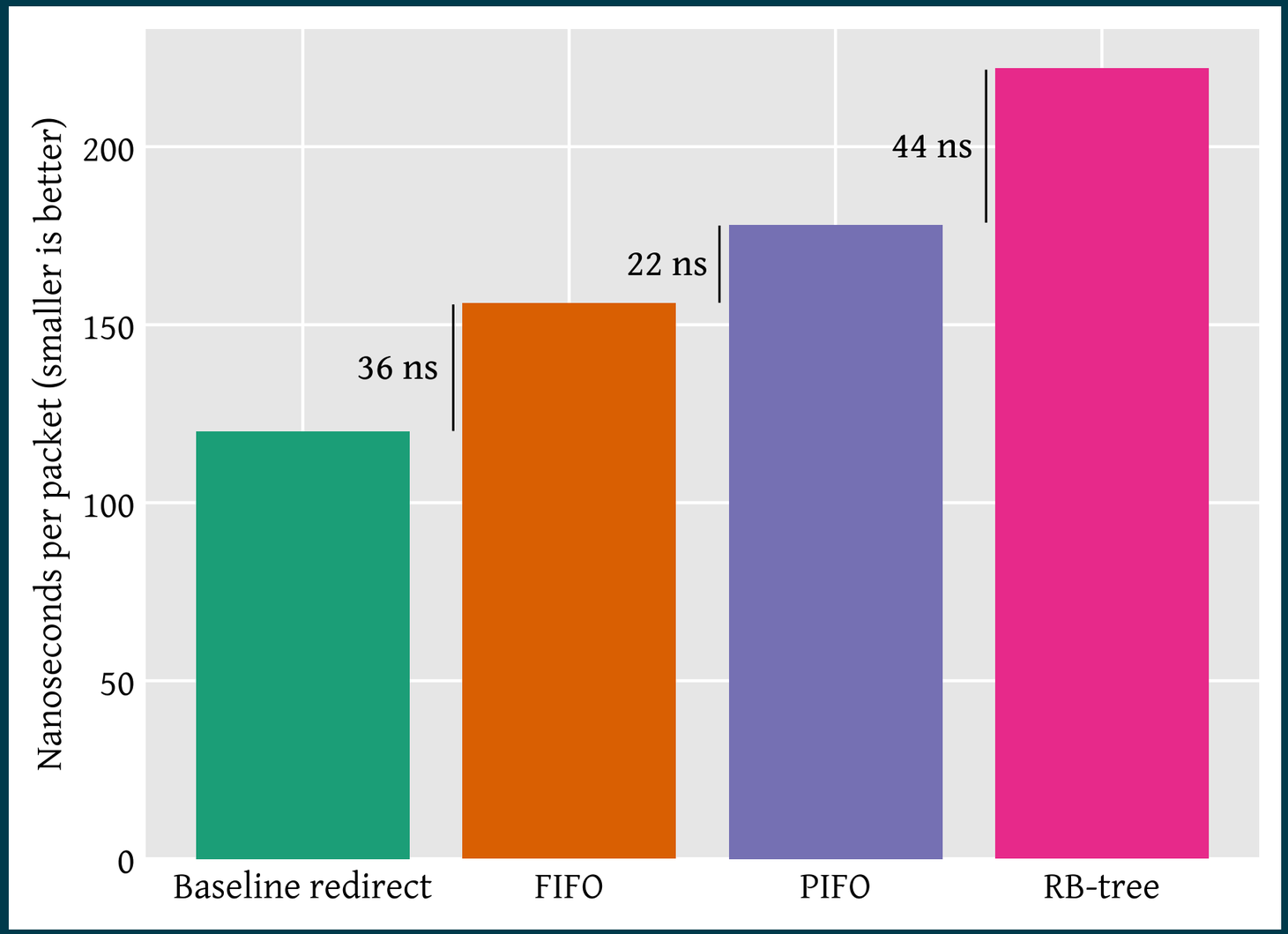


Figure 4 from Saeed et al (2019)

Performance overhead of map types (again)



Data structures: Summary

- We need a data structure (BPF map) to store packets
  - Current qdiscs only use two data structures: FIFO and priority queue
  - A priority queue can be used as a FIFO, so really only one
- API: `bpf_redirect_map()` to enqueue, add `bpf_packet_dequeue()`
- The Eiffel PIFO algorithm performs well
  - Drawback: Priority range is fixed / only growing
  - Is this API limitation acceptable?

Recall: The ingredients we need

- Somewhere to store packets
- A way to schedule dequeue and transmission

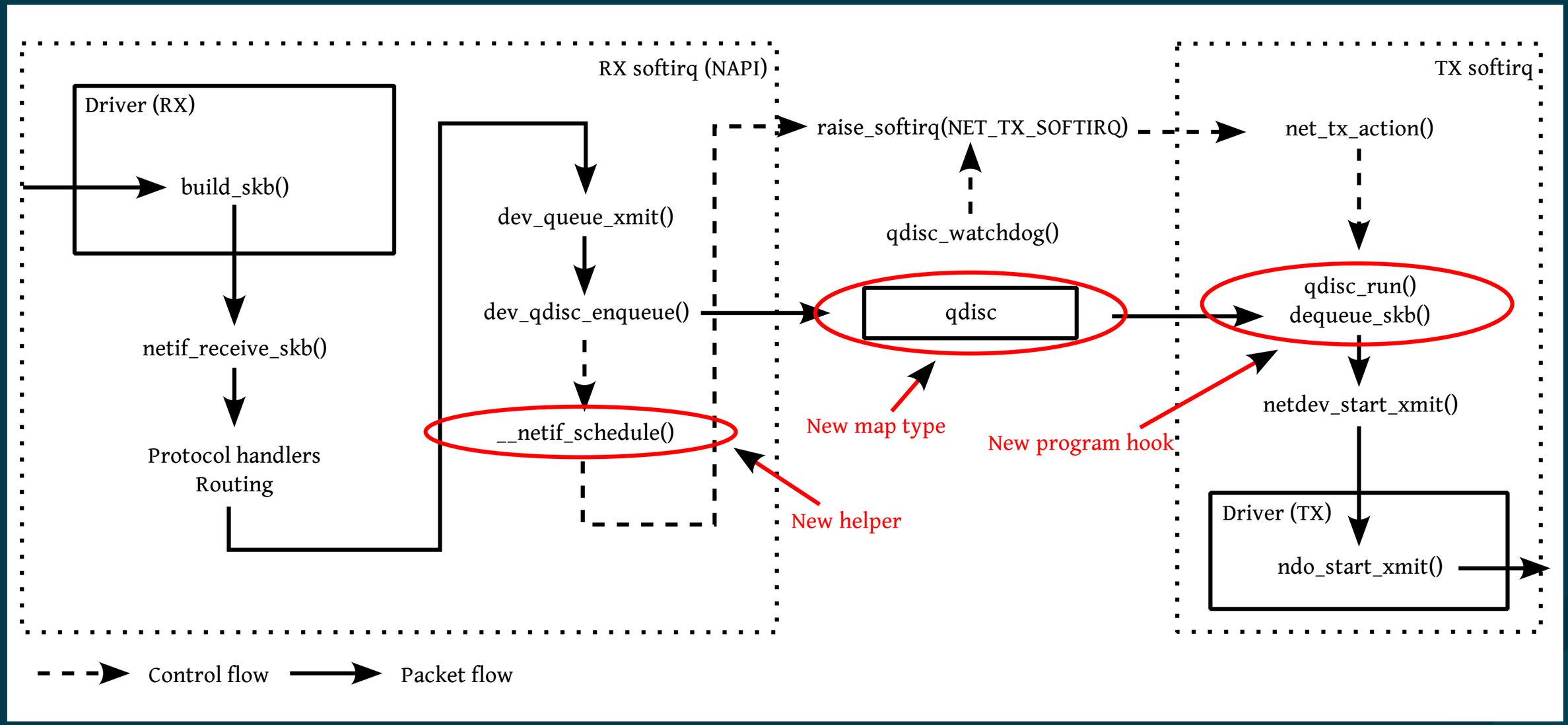
TX hook attempt 1: dequeue hook

New xdp dequeue program type

- Can be attached to an interface (like XDP program)
- Returns a packet to transmit
- Stack calls `ndo_xdp_xmit()` with batch of packets

Submitted as RFC series: https://lore.kernel.org/r/20220713111430.134810-1-toke@redhat.com

TX scheduling attempt 1 – replicate netstack



TX hook attempt 1: example code

```
struct pifo_map {
        __uint(type, BPF_MAP_TYPE_PIFO_XDP);
        __uint(key_size, sizeof(__u32));
        __uint(value_size, sizeof(__u32));
        __uint(max_entries, 10240);
        __uint(map_extra, 8192); /* range */
} pifo SEC(".maps");

SEC("xdp")
int xdp_redirect_map_queue(struct xdp_md *ctx)
{
        int ret;
        ret = bpf_redirect_map(&pifo, 0, 0);

        if (ret == XDP_REDIRECT)
                bpf_schedule_iface_dequeue(ctx,
                                           tgt_ifindex,
                                           0);

        return ret;
}
```

```
SEC("xdp_dequeue")
void *xdp_redirect_deq_func(struct dequeue_ctx *ctx)
{
        struct xdp_md *pkt;
        __u64 prio = 0;

        pkt = (void *)bpf_packet_dequeue(ctx, &pifo,
                                         0, &prio);
        if (!pkt)
                return NULL;

        return pkt;
}
```

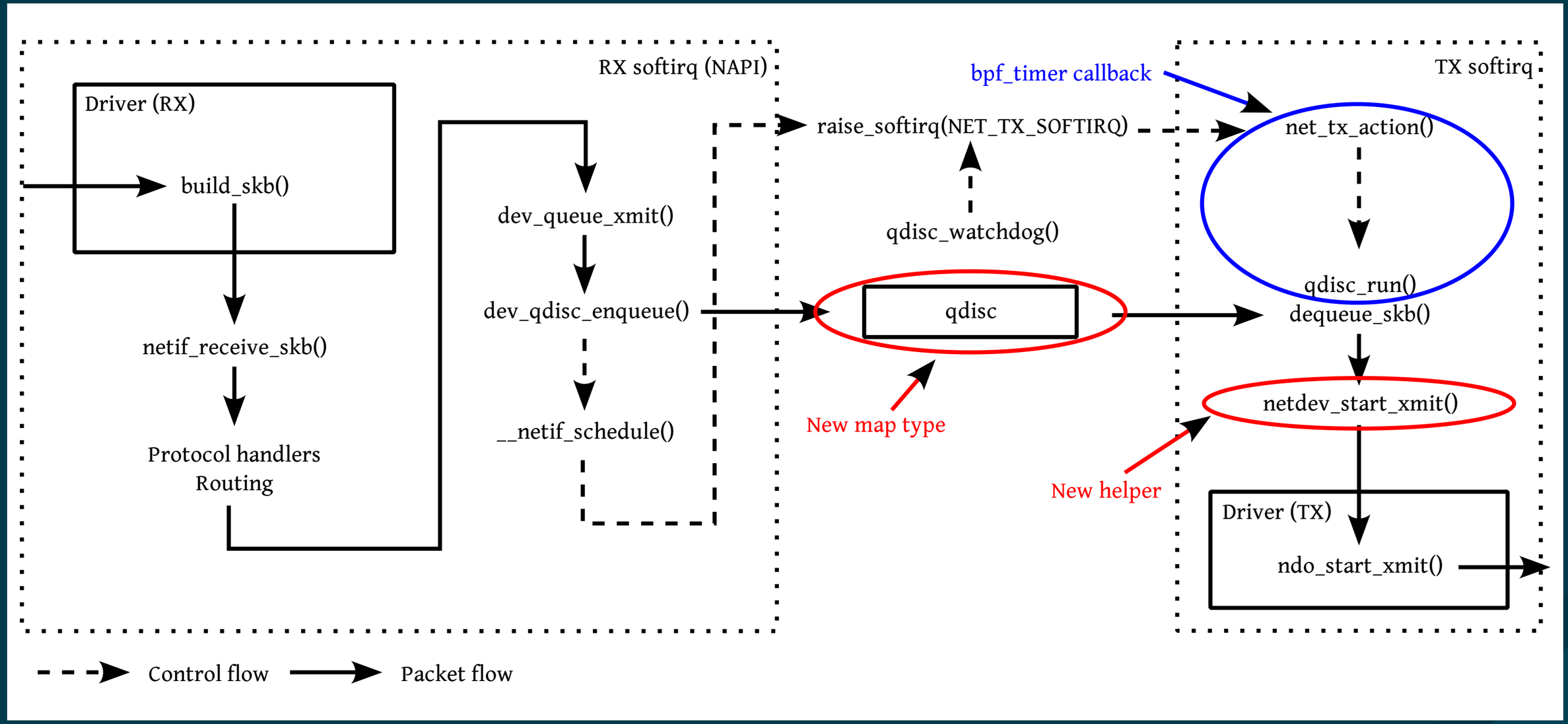
TX hook attempt 1: Problems

Problem: The maintainers didn't like it

> *This feature can be done similar to hid-bpf without cast-in-stone uapi and hooks. Such patches would be much easier to land and iterate on top. The amount of bike shedding will be 10 times less. No need for new program type, no new hooks, no new FDs and attach uapi-s.*
>
> *Alexei in https://lore.kernel.org/r/20220715011228.tujkugafv6eixbyz@MacBook-Pro-3.local*

Turns out he was (almost) right! As seen by attempt 2...

TX hook attempt 2: Use bpf_timers

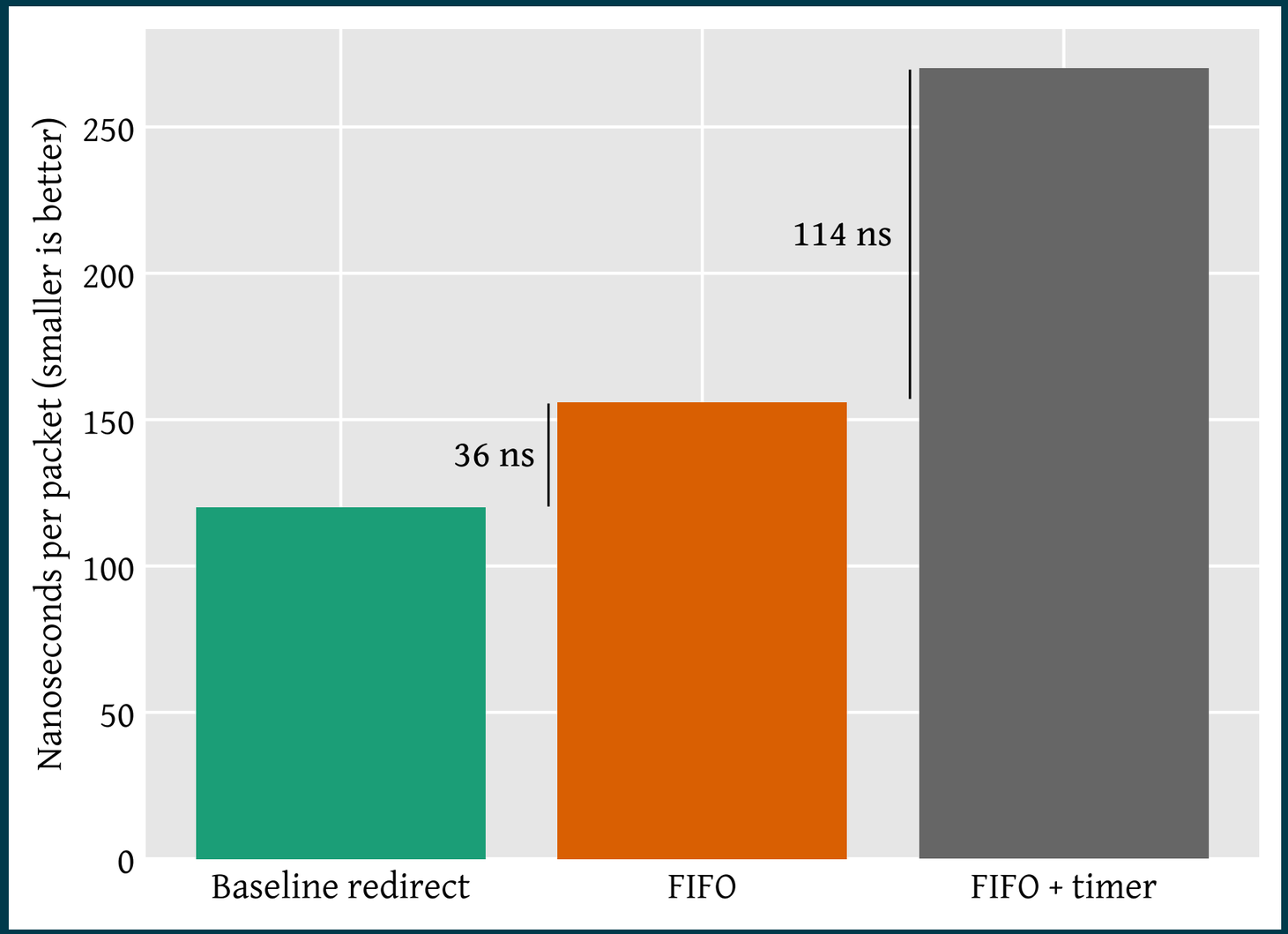


TX hook attempt 2: Example code

```c
__u64 num_queued = 0;

SEC("xdp")
int xdp_redirect_map_timer(struct xdp_md *ctx)
{
        struct bpf_timer *timer;
        int ret, array_key = 0;

        timer = bpf_map_lookup_elem(&timermap,
                                    &array_key);
        if (!timer)
                return XDP_ABORTED;

        if (!timer_init) {
                bpf_timer_init(timer, &timermap,
                               CLOCK_MONOTONIC);
                bpf_timer_set_callback(timer,
                                       xdp_timer_cb);
                timer_init = 1;
        }

        ret = bpf_redirect_map(&pifo, 0, 0);
        if (ret == XDP_REDIRECT) {
                num_queued++;
                bpf_timer_start(timer,
                                0 /* call asap */, 0);
        }
        return ret;
}
```

```c
#define BATCH_SIZE 128

static int xdp_timer_cb(void *map, int *key,
                        struct bpf_timer *timer)
{
        struct xdp_md *pkt;
        __u64 prio = 0;
        int i;

        for (i = 0; i < BATCH_SIZE; i++) {
                pkt = (void *)bpf_packet_dequeue_xdp(&pifo,
                                                     0,
                                                     &prio);
                if (!pkt)
                        break;

                num_queued--;
                bpf_packet_send(pkt, tgt_ifindex, 0);
        }

        bpf_packet_flush();
        if (num_queued)
                bpf_timer_start(timer,
                                0 /* call asap */, 0);

        return 0;
}
```

Problem: Overhead of bpf_timer

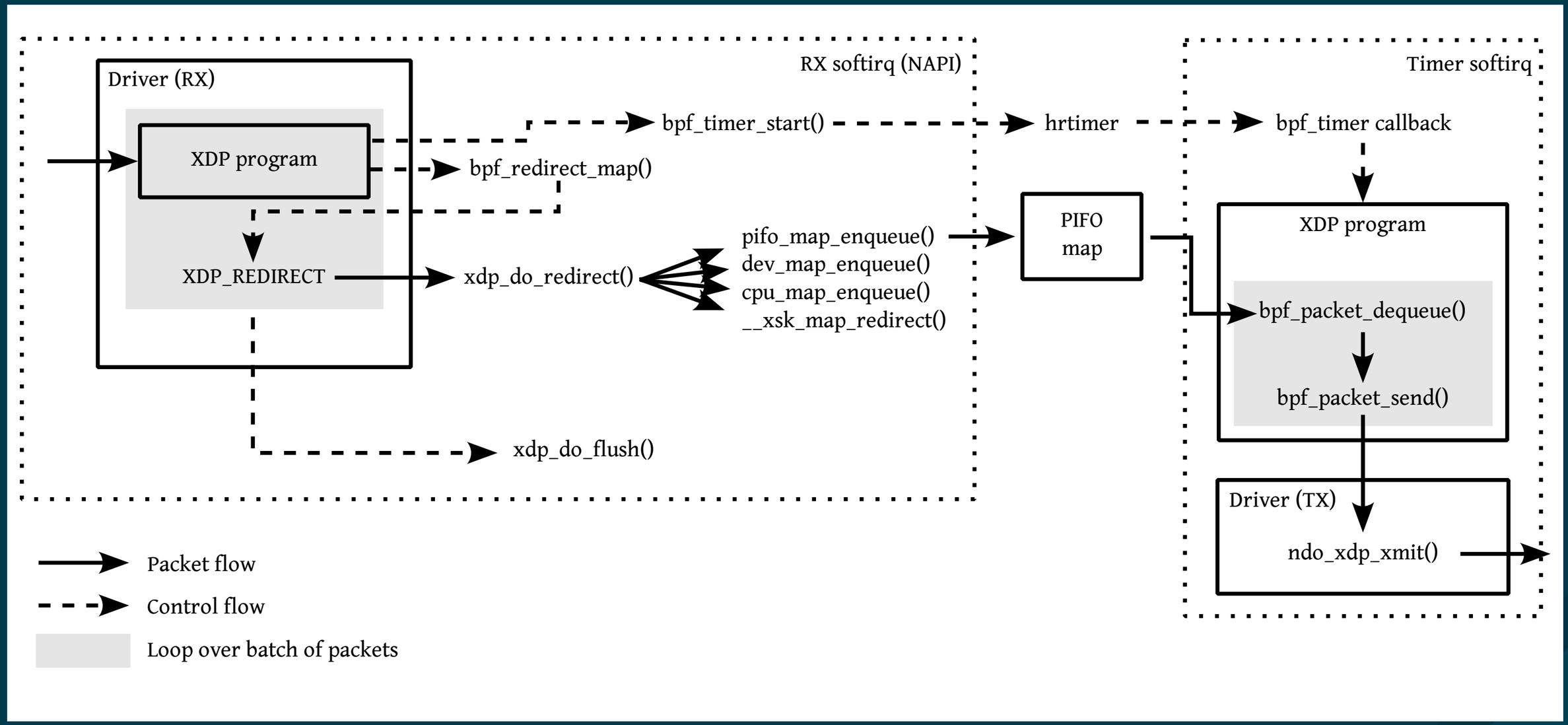


TX hook attempt 2: Problems

The callback approach seems promising, but has a few problems:

- Performance of `bpf_timer`
  - Overhead (previous slide)
  - Risk of blocking other timer work (?)
  - Replace by generic callback feature as discussed in https://lore.kernel.org/r/cover.1657576063.git.delyank@fb.com?
- No pushback from driver
  - How does the BPF program know that the interface is busy ?
  - With TX hook stack can keep packets around, what does BPF do?

Summary: XDP queueing design



End: Questions?

WiP code:

- Kernel patches (implementing both TX hook approaches): https://git.kernel.org/toke/l/xdp-queueing-07
- Test framework for queueing algorithms: https://github.com/xdp-project/bpf-examples/pull/40 (by my PhD student Freysteinn Alfredsson)

Many thanks to Kumar Kartikeya Dwivedi, Jesper Brouer, Anna Brunstrom and Per Hurtig, as well as everyone who reviewed the RFC patchset.

Bonus slide: BPF qdisc

There's a separate BPF qdisc proposal being worked on by Cong Wang.

Latest RFC: https://lore.kernel.org/r/20220602041028.95124-1-xiyou.wangcong@gmail.com

This is complementary to queueing in XDP - not in competition.

- BPF qdisc for packets going through the stack, XDP queueing is for bypassing the stack when forwarding
- Can hopefully share BPF map type and helpers
- BPF code reuse will likely be similar to TC-BPF/XDP (i.e., some effort required)

Bonus slide: CPU steering

For good forwarding performance, splitting work among CPUs is essential.

- For XDP, this is up to the BPF program.
  - All callbacks will be on the same CPU
  - Steering can be done today using cpumap, see: https://github.com/xdp-project/xdp-cpumap-tc
- Possible optimisation: Bind map to particular CPU to elide locking