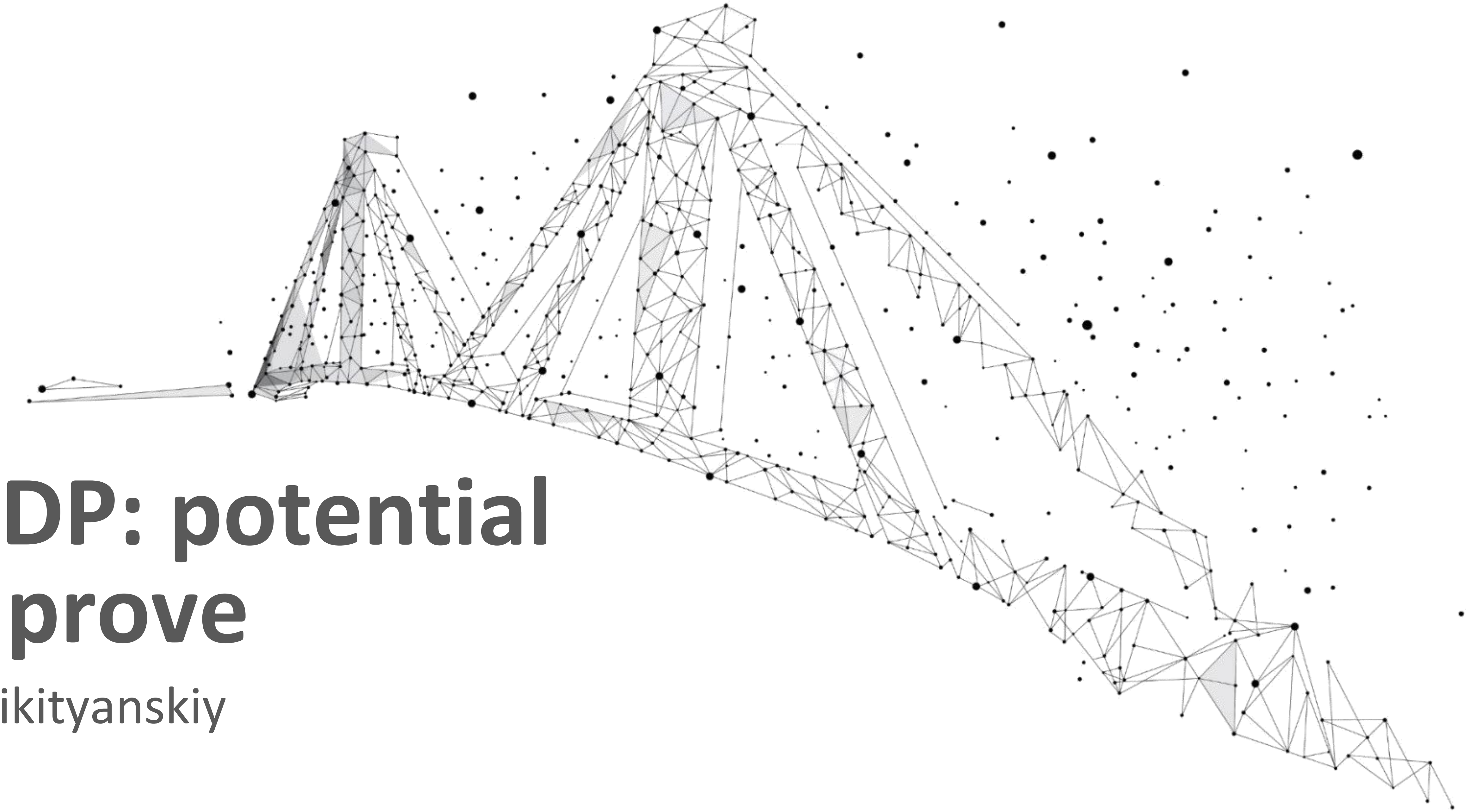


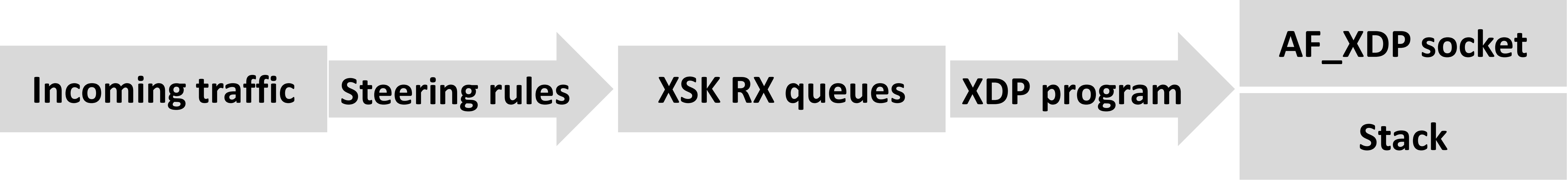
AF_XDP: potential to improve

Maxim Mikityanskiy

June 2019

Queues

XSK RX queue



- XSK RX queue != regular RX queue

	Regular RX queue	XSK RX queue
Memory model	Dynamic allocation	Allocation from UMEM
XSK RX	Extra copy	Zero-copy
XDP_DROP	Fast	Fast
XDP_PASS	Build XSK in place	Extra copy*
XDP_TX	Page reuse	Extra copy*
XDP_REDIRECT	Page reuse	Extra copy

*Can be potentially improved.

XSK RX queue allocation scheme

- Replacing regular RX queues by XSK ones – disadvantages:
  - Same index range – no way to distinguish.
  - Opening an XSK breaks the regular traffic flow because of RSS.
  - RSS management is not easy.
  - Opening an XSK requires restarting a channel.
- XSK RX queues should be a separate queue type.
  - Own numeration.
  - Opening dedicated XSK RX queues in existing channels.
  - Allocating additional XSK RX queues.
- XSK RX queues are to be registered in the kernel.
  - Attach UMEM.
  - More on this later.

Queues or channels?

- XSK is both RX and TX, but it has only a single queue index.
- libbpf’s xsk_get_max_queues() queries the number of combined channels.
  - It doesn’t correspond to what the kernel does.
- Everything looks like it’s designed to be used with combined channels, but instead the netdev queues are used, and they don’t fit well.

```
struct xdp_umem *xdp_get_umem_from_qid(struct net_device *dev,
                                       u16 queue_id)
{
        if (queue_id < dev->real_num_rx_queues)
                return dev->_rx[queue_id].umem;
        if (queue_id < dev->real_num_tx_queues)
                return dev->_tx[queue_id].umem;

        return NULL;
}
```

- There is a relation between RQ #X and SQ #X, so the abstraction of a combined channel is natural.
- Proposal: fix the terminology and switch to using channel ID instead of QID.

The way to register XSKs in the kernel

- A combined channel in the driver consists of:
  - Regular RQ and SQ.
  - XDP SQs.
  - XSK RQ and SQ created on demand.
- struct net_device will have an array of XSK QP structs.
- UMEMs for non-zero-copy mode are to be stored in regular queues.
- XSK QPs correspond to XSK RQ and SQ of a channel in the driver.
- Unbound XSK QPs.
  - A suggestion in Magnus's RFC: https://patchwork.ozlabs.org/cover/1094083/.
  - With an XSK QP as a separate entity, it's easy to allocate new QPs on demand.
    - Dedicated NAPI.
    - Not bound to an IRQ.
    - Not bound to a channel.

Speeding up slow path

Zero-copy XDP_TX and XDP_PASS

- Jonathan Lemon had a PoC patch that implements zero-copy XDP_TX.
  - The frame is put to the Reuse Ring once the TX completes.
  - The issue is that the Reuse Ring can overflow.
- Keep UMEM frames in the driver – options:
  - Bigger Reuse Ring with a fallback to copy.
  - Return these frames to the Return Ring in the application.
  - Return to the Completion Ring – if the application supports.
- XDP_PASS issue: userspace has write access to the UMEM, kernel parsers can be confused.

Return Ring

Return Ring

- Example use cases:
  - Return frames on shutdown.
  - Return frames which are not XDP_REDIRECTed to an XSKMAP.
  - Signal about the empty Fill Ring.
- Descriptor:
  - Error code.
  - Frame handle.

Corner cases

- Frames are owned by the driver, but the interface goes down.
  - Reuse Ring as a workaround: https://patchwork.ozlabs.org/patch/962914/#1982161.
  - TX frames are completed without transmission and error indication.
  - Lack of a common cleanup mechanism in the kernel.
  - Return Ring to solve the problems.
- XDP program doesn’t return XDP_REDIRECT to an XSKMAP.
  - Recycle internally.
  - Lack of a standard way.
  - Reuse Ring can be used.
    - Interferes with zero-copy XDP_TX.
  - Return Ring can be used.
    - XDP_PASS is faster. Is it a real use case?
    - A roundtrip through the userspace slows things down.
  - Use the Reuse Ring while possible; on shutdown flush to the Return Ring.
- An abstraction layer over the Reuse and Return Rings.
  - Provide a common algorithm to all drivers.

Corner cases

- TX packet size > MTU.
  - No error reporting — a completion is simply issued.
    - AF_PACKET returns –EMSGSIZE.
  - Requires some manipulations to issue completions in order.
- TX completes with an error.
  - Driver can try to recover transparently.
    - Is it driver’s responsibility?
    - Most likely, retrying will lead to the same error.
  - If the recovery is impossible, tell the application.

Corner cases

- XDP program increases the packet size over MTU. Should we pass it to AF_XDP?
  - Depends on the use case.
    - Application receives a packet bigger than MTU and tries to respond with a packet that big.
    - Application implements a custom stack, which drops oversized packets.
  - Suggestion: to drop oversized packets, unless they go to AF_XDP.

Lack of notification mechanism

- Addressed by a recent series by Magnus: https://patchwork.ozlabs.org/cover/1115314/.
- Busy-polling on RX
  - If the application stops refilling the Fill Ring, NAPI busy polls.
- Busy-polling on TX
  - The driver doesn’t guarantee that it consumes everything for transmission on sendto().
  - The application has to grind CPU with syscalls.
- Unresolved issue?
  - xdpsock in poll mode can get stuck if the TX Ring is full, and nothing is sent on the only sendto() call.

Configuration

Steering to XSK queues

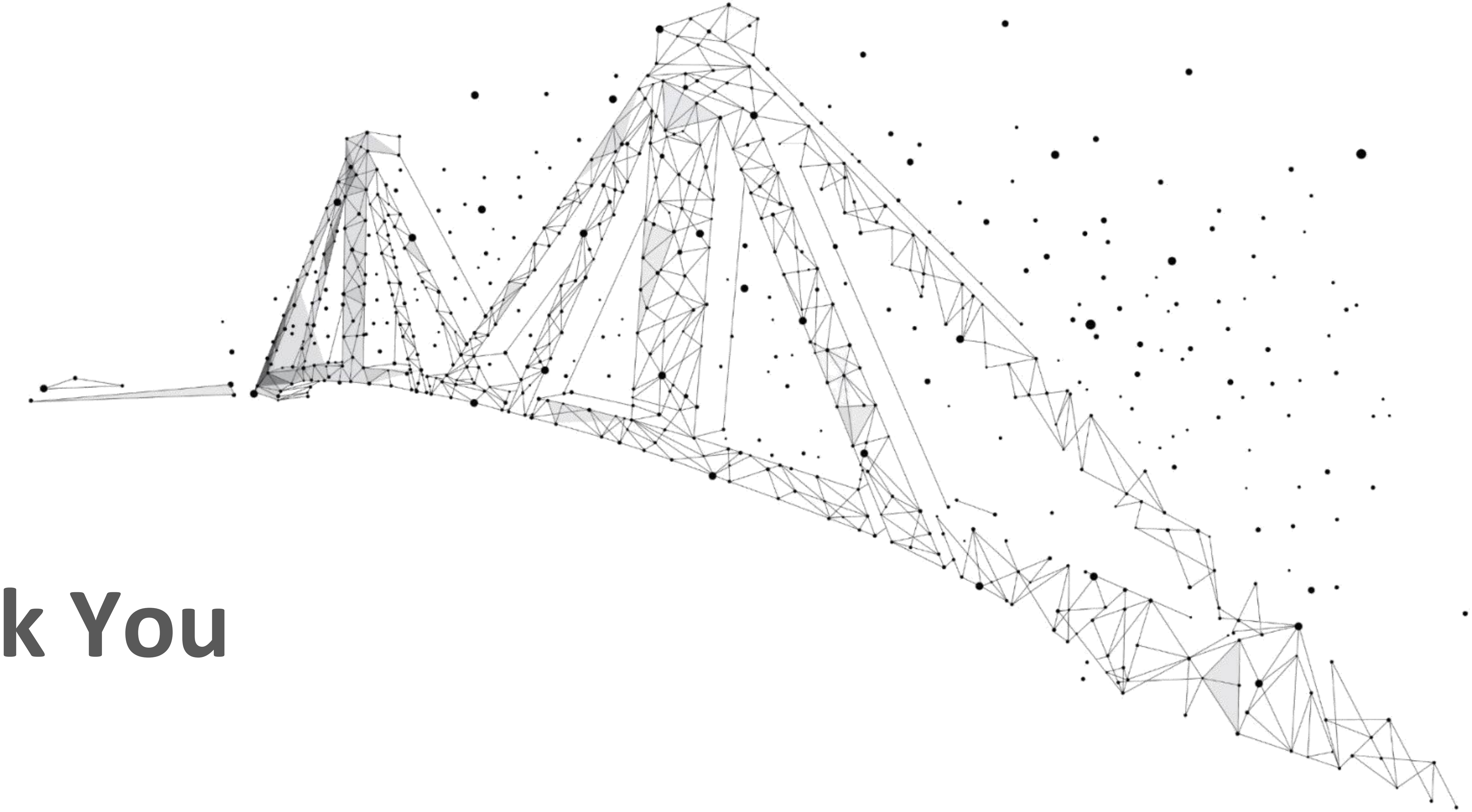
- API needs to be extended:
  - Allow to choose XSK/regular RX queue.
  - Allow to steer traffic to unbound XSKs.
- Use tc flower instead of ethtool?

Non-ZC fallback

- If XSK queue X is requested, but the driver is non-ZC, fall back to regular RX queue X.
- Problematic with unbound XSK QPs.
- Different steering configuration for ZC and non-ZC.
  - The driver can ignore is_xsk.
- XDP program works differently in the compatibility mode.
  - The configuration can be passed through a BPF map.
  - Different programs can be loaded.

RSS for XSK queues

- A real use case.
- Mellanox hardware supports it.
- Lack of software interface to configure it.
- A rejected series by Edward Cree: https://patchwork.ozlabs.org/cover/878725/.
    - Reimplement it with tc flower?



Thank You