Extending AF_XDP with hardware metadata

Stanislav Fomichev, Google, 2023

Problem Statement

Expose NIC hardware offload capabilities to AF_XDP.

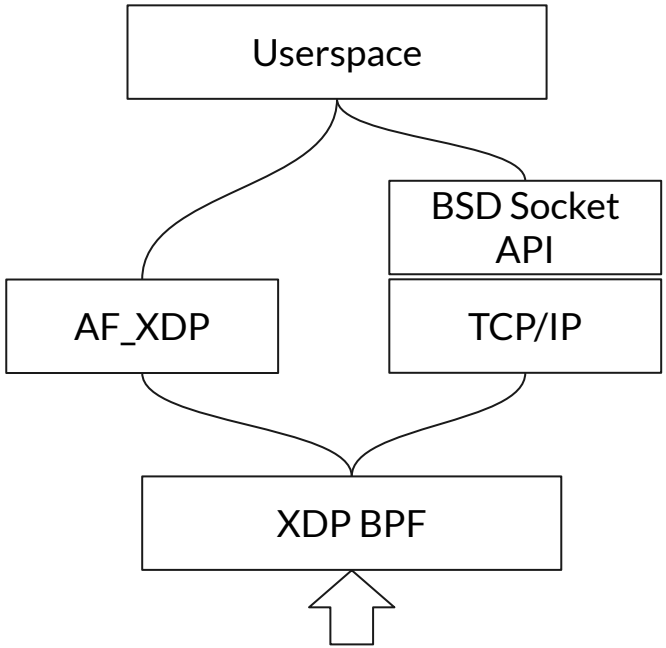
Short AF_XDP introduction

- Protocol family to support sending / receiving raw frames from the netdev
  - socket(AF_XDP, ...)
- A modern version of the AF_PACKET, man 7 packet
- Plays nicely with the kernel, doesn't take over the interface
- Designed to be zerocopy-friendly
- I haven't invented it
  - Björn Töpel
  - Magnus Karlsson
  - and many more
- Documentation/networking/af_xdp.rst

AF_XDP vs XDP

- AF_XDP is not to be confused with the XDP
- XDP is only to steer the frames into AF_XDP consumer
- XSK == AF_XDP

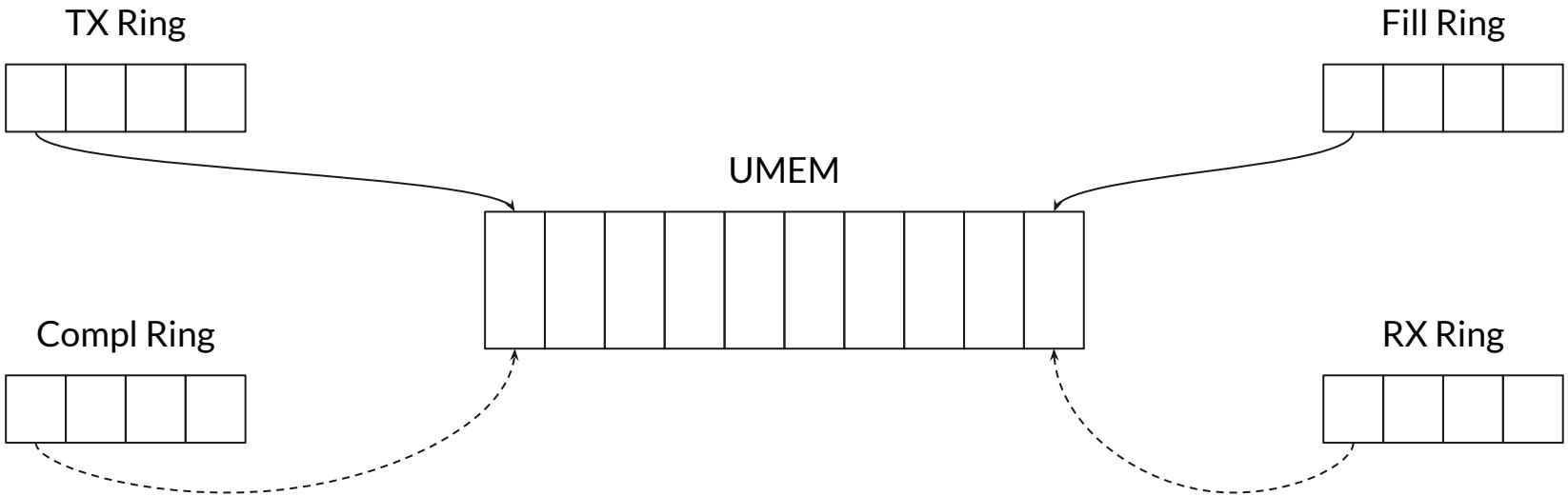
*in theory, if HW supports n-tuple flow steering, there might be a case to support XDP-less AF_XDP where AF_XDP would take over one or more separate HW queues



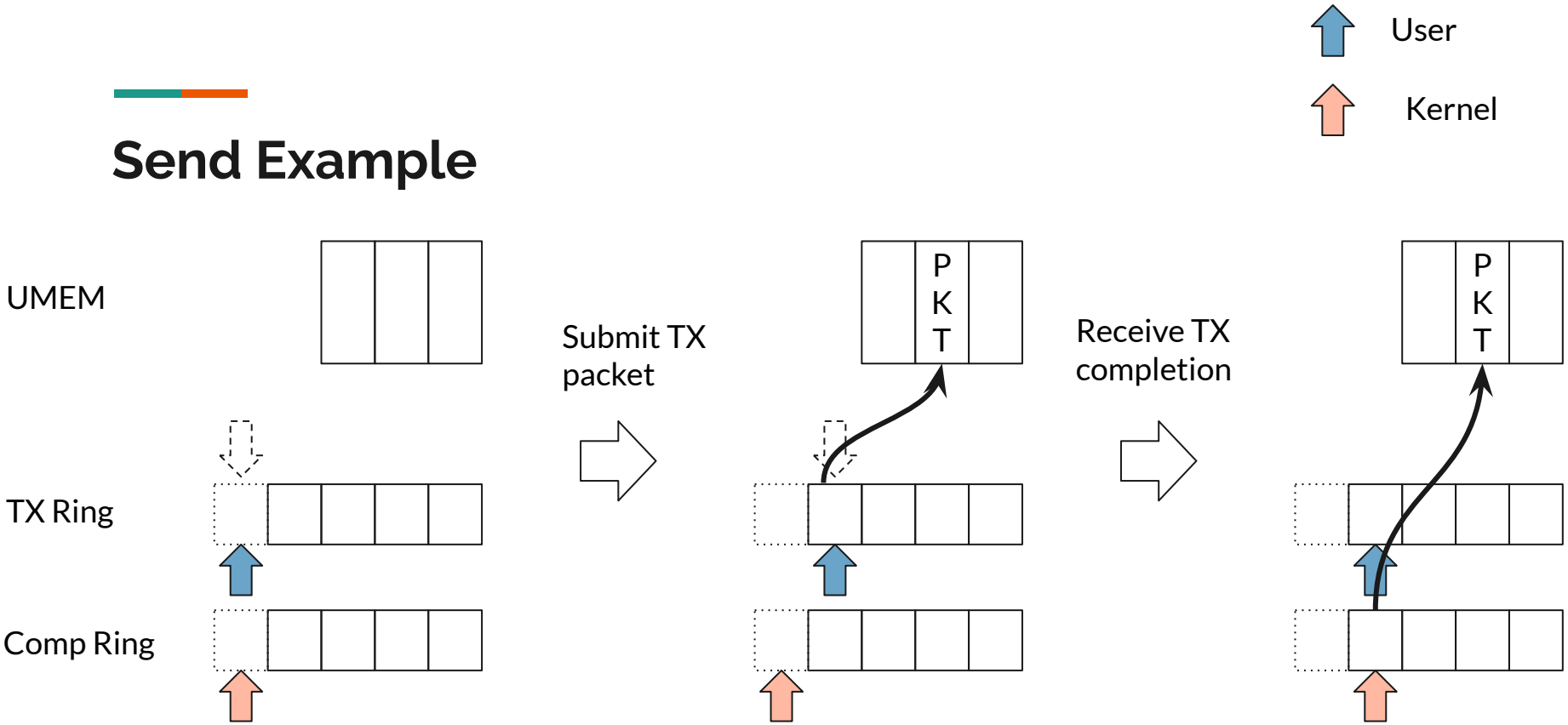
What about DPDK?

- DPDK is more invasive, "kernel bypass"
  - DPDK exposes real/raw hardware queue
  - DPDK exposes HW offloading capabilities
  - AF_XDP abstracts actual queue implementation
- DPDK reimplements existing kernel drivers
  - Actually can use AF_XDP to send/receive the frames
- DPDK uses spinning mode (need to dedicate processing cores)
- DPDK still (slightly) faster than AF_XDP

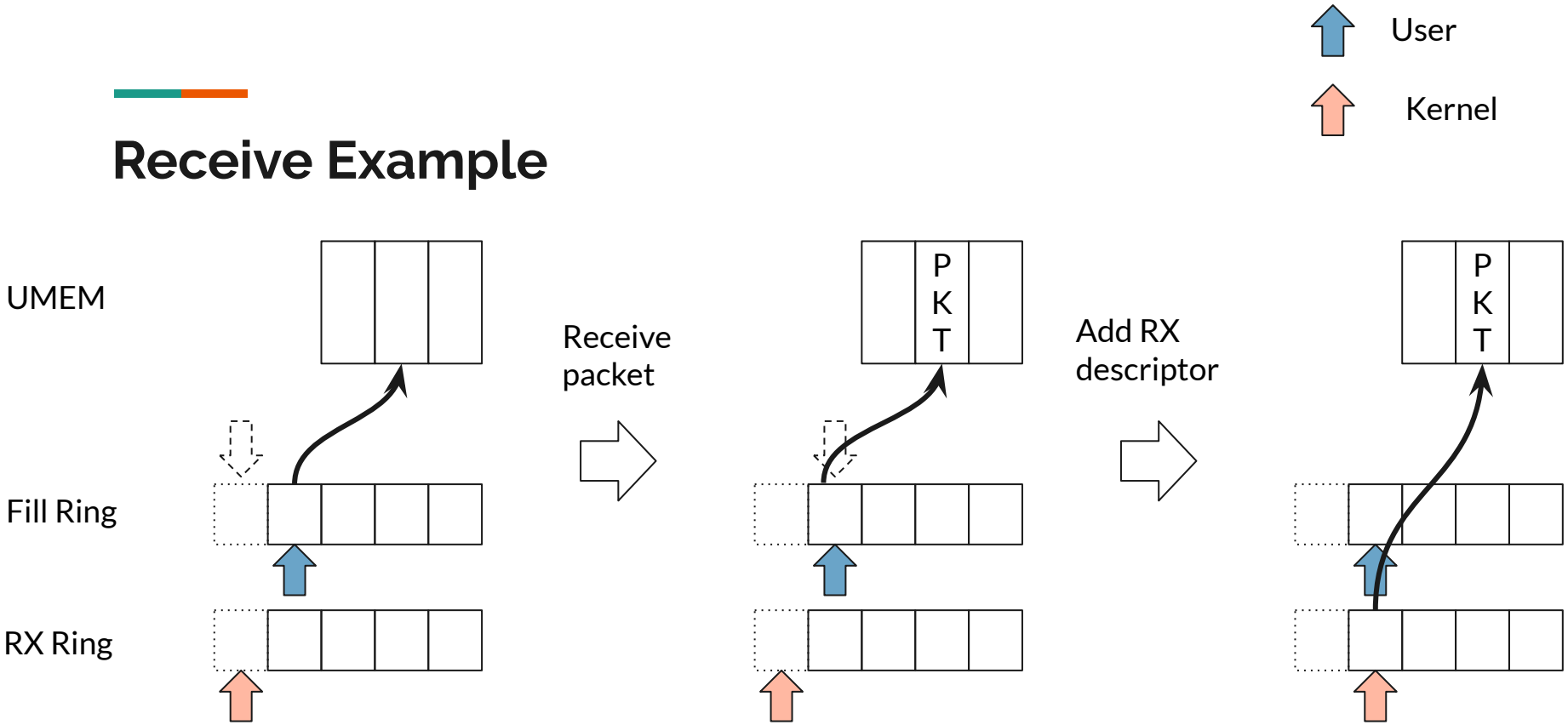
AF_XDP Rings



Send Example



Receive Example



Receive Steering

```
struct {
        __uint(type, BPF_MAP_TYPE_XSKMAP);
        __uint(max_entries, 4);
        __type(key, __u32);
        __type(value, __u32);
} xsk SEC(".maps");

SEC("xdp")
int rx(struct xdp_md *ctx) {
        // filter packets into AF_XDP (xsk) based on something
        return bpf_redirect_map(&xsk, ctx->rx_queue_index, XDP_PASS);
}
```

What's missing?

Now that we're done in the intro, let's discuss what we've added to AF_XDP.

Metadata? Hints? Offloads?

- We need some way to consume and communicate some meta information about the packet
  - We want to fully utilize NIC HW offload capabilities
- Examples are, on RX
  - NIC has verified L4 (TCP/UDP) checksum
  - NIC has computed flow hash over the L3 and/or L4
  - Nic wants to communicate HW receive timestamp
- On TX
  - We want to ask NIC to calculate L4 checksum starting at offset
  - We want to receive HW TX timestamp
- Don't have a solid name for this, have been using metadata/hints/offloads

Recent AF_XDP Limitations

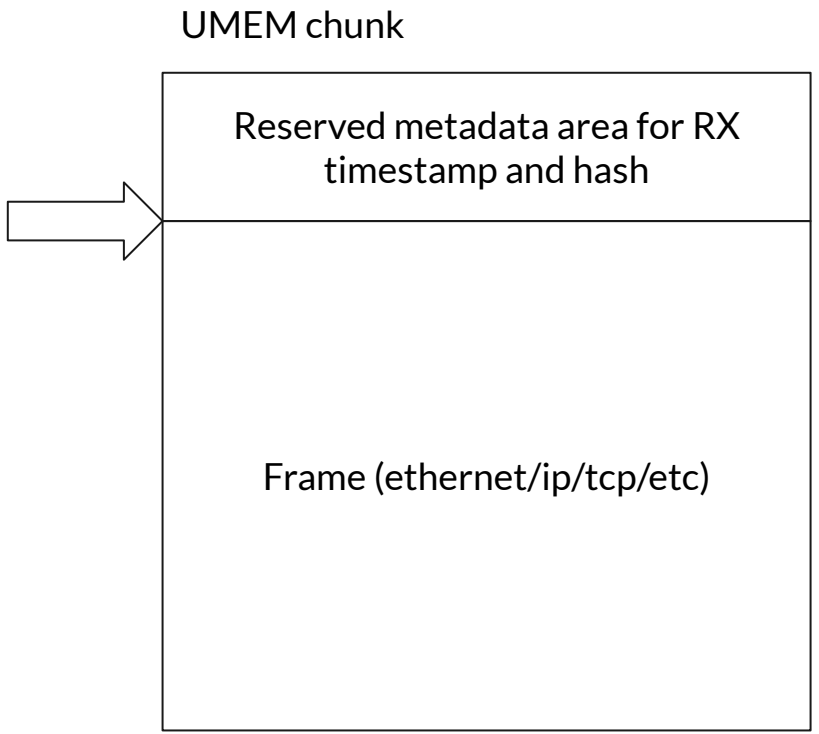
- MTU [solved, Linux 6.6]
  - 4k page limit
- Scatter Gather [solved, Linux 6.6]
  - everything had to go via single linear buffer
- No access to HW offloads on receive [solved, Linux 6.3]
- No access to signal HW offloads on transmit [in progress, hopefully in Linux 6.8?]

Receive Side

- Added the ability to read HW offload information from XDP program
- Set of pre-defined BPF kfuncs
  - bpf_xdp_metadata_rx_timestamp
  - bpf_xdp_metadata_rx_hash
- The output of those kfuncs can be put into "metadata" are in the AF_XDP umem chunk
  - the layout of this metadata is flexible and its up to the BPF program and AF_XDP consumer to agree on the layout

Receive Side

Fixed offset that has been agreed upon between BPF program and AF_XDP consumer



Receive Side Code Sample

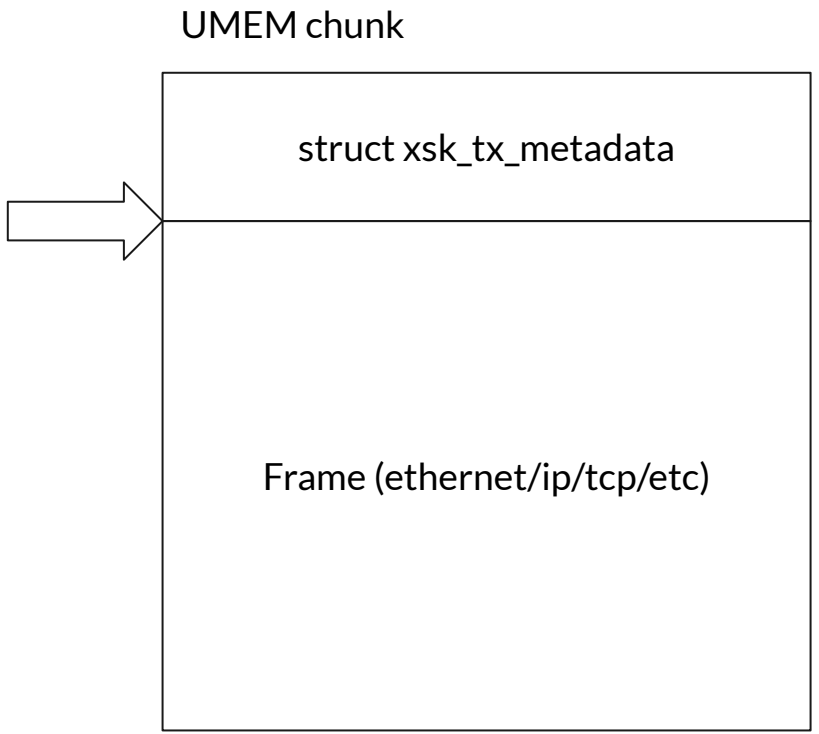
```
SEC("xdp")
int rx(struct xdp_md *ctx) {
      bpf_xdp_adjust_meta(-sizeof(u64));
      bpf_xdp_metadata_rx_timestamp(ctx, &ctx->data_meta);
      return bpf_redirect_map(&xsk, ctx->rx_queue_index, XDP_PASS);
}

// in userspace AF_XDP consumer, when the frame is received
payload = xsk_umem__get_data(...);
__u64 *timestamp = payload - sizeof(u64);
```

Transmit Side

- We don't have XDP on TX, so the approach is less flexible
  - My original attempt was to add some light-weight XDP alternative at TX, but it wasn't well received
- Have a fixed metadata layout between the kernel and AF_XDP producer

Transmit Side

The caller indicates that it intends to use the metadata via UMEM config.



Transmit Side Code Sample

```
// in userspace AF_XDP producer
payload = xsk_umem__get_data(...);
struct xsk_tx_metadata *meta = payload - sizeof(struct xsk_tx_metadata);

// request checksum offload
meta->request.flags |= XDP_TXMD_FLAGS_CHECKSUM;
meta->request.csum_start = sizeof(*eth) + sizeof(*iph);
meta->request.csum_offset = offsetof(struct udphdr, check);
```

Current Status

RX https://lore.kernel.org/bpf/20230119221536.3349901-1-sdf@google.com/

- Access to receive hardware timestamp
- Access to receive hash

TX https://lore.kernel.org/bpf/20231102225837.1141915-1-sdf@google.com/T/#t

- Access to transmit completion hardware timestamp
- Support for TCP/UDP transmit checksum offload

Both RX and TX are extensible, so more to come!

Future Receive Side

- Things that have been flowing through the list so far
- VLAN (Larysa Zaremba)
- RX checksum (Larysa Zaremba)
    - dropped from the latest series to make it easier to push the rest
    - hopefully can follow up after VLAN is in

Future Transmit Side

- TX departure-time - have patches from Intel's Song Yoong Siang
- TX crypto offloads
  - PSP?
- TSO and USO
  - I've seen on the least concerns about userspace TCP

Questions?

Thank you to all upstream reviewers for feedback and guidance!