ADDING AF_XDP ZERO-COPY SUPPORT TO DRIVERS

Maxim Mikityanskiy, 2020

AGENDA

AF_XDP 101

Basics that you need to know before implementing AF_XDP support in the driver

Driver implementation

All required details to add AF_XDP zero-copy support to the driver

Extra stuff

More advanced features: unaligned chunks, need_wakeup

AF_XDP-related challenges

PREREQUISITES

XDP support in the driver

Basic knowledge about AF_XDP

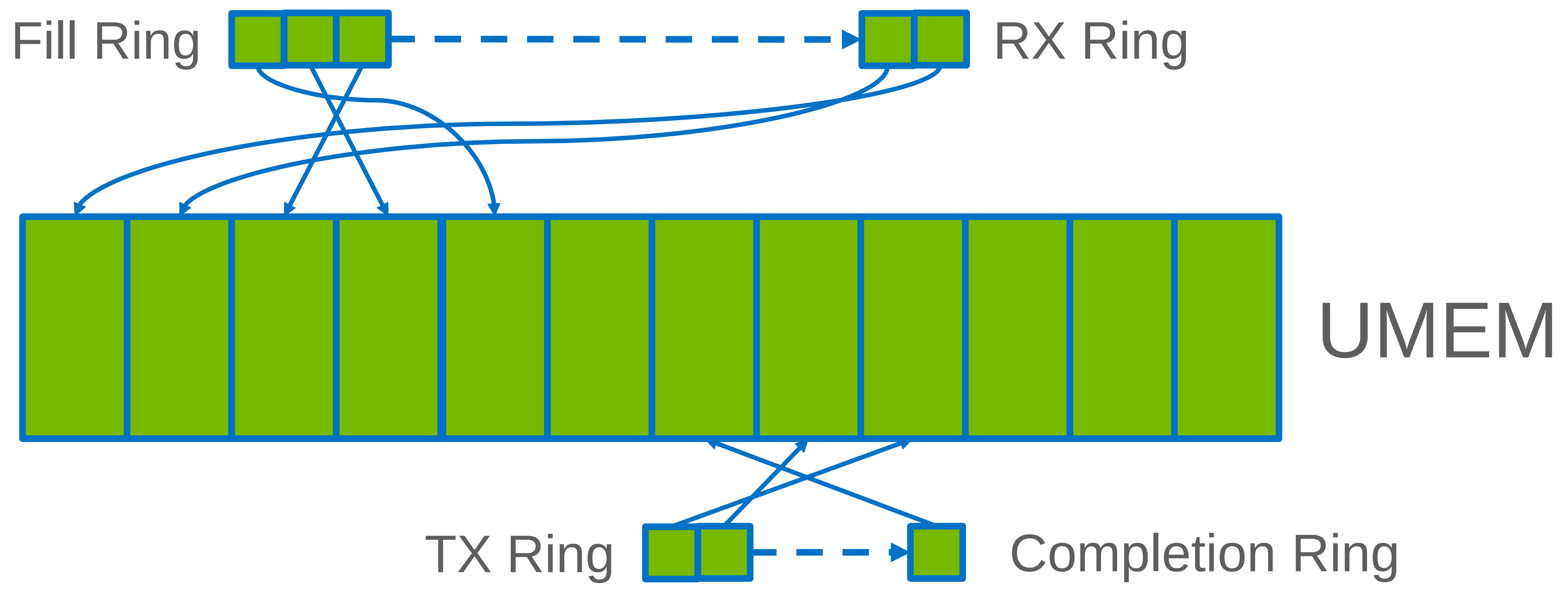
AF_XDP 101

AF_XDP 101 – UMEM

Allocated by the application using `mmap()`

Shared between the application, driver and hardware

Bidirectional DMA mapping



AF_XDP 101

Driver support is needed for zero-copy

- AF_XDP has (slow) fallbacks when zero-copy or XDP are not supported

Most of the internals are encapsulated in AF_XDP core

Minimal interface is exposed to the driver

DRIVER IMPLEMENTATION

WHAT YOU’RE GOING TO DO IN THE DRIVER

- Support XDP
- Implement NDOs (setup, wakeup)
- When XSK is enabled, create XSK RX queue and TX queue
- For XSK RX queue, allocate buffers from UMEM
- Handle XSK TX in NAPI

WHY WE NEED SEPARATE QUEUES FOR XSK

XSK RX: allocation of buffers happens from the UMEM

XSK TX: simplify teardown

KERNEL API

```
// Setup
.ndo_bpf
xdp_rxq_info_reg_mem_model
xdp_get_umem_from_qid
xsk_umem_get_headroom
xsk_umem_get_chunk_size
xsk_buff_set_rxq_info
xsk_buff_dma_map
xsk_buff_dma_unmap

// need_wakeup
xsk_umem_uses_need_wakeup
xsk_set_rx_need_wakeup
xsk_clear_rx_need_wakeup
xsk_set_tx_need_wakeup
xsk_clear_tx_need_wakeup
```

```
// TX/RX
.ndo_xsk_wakeup

// RX
xsk_buff_can_alloc
xsk_buff_alloc
xsk_buff_xdp_get_frame_dma
xsk_buff_dma_sync_for_cpu
xsk_buff_free

// TX
xsk_umem_consume_tx
xsk_buff_raw_get_dma
xsk_buff_raw_get_data
xsk_buff_raw_dma_sync_for_device
xsk_umem_consume_tx_done
xsk_umem_complete_tx
```

NDO STUBS

```
// .ndo_bpf:
switch (xdp->command) {
case XDP_SETUP_XSK_UMEM:
        return vnd_xsk_setup_umem(dev, xdp->xsk.umem, xdp->xsk.queue_id);
}

// .ndo_xsk_wakeup:
int vnd_xsk_wakeup(struct net_device *dev, u32 qid, u32 flags)
{
        // Check that XDP program is set, and XSK is enabled on queue qid.
        // Return an error otherwise.

        if (!napi_if_scheduled_mark_missed(napi))
                // Trigger an IRQ using hardware mechanisms.

        return 0;
}
```

SETUP STAGE

```
int vnd_xsk_setup_umem(struct net_device *dev, struct xdp_umem *umem, u16 qid)
{
        return umem ? vnd_xsk_enable_umem(dev, umem, qid) :
                      vnd_xsk_disable_umem(dev, qid);
}
```

The kernel makes sure no UMEM is attached to this queue when it enables XSK.

You'll need to keep track which queues have a UMEM attached:

- Store UMEM pointers yourself.
- Store a flag and acquire the UMEM pointer by `xdp_get_umem_from_qid`.
  - You need this flag to distinguish from non-zero-copy AF_XDP.

You need to store it where it survives reset of queues. When the netdev is brought up, it will be possible to determine if a queue is XSK-enabled.

SETUP STAGE

```
int vnd_xsk_enable_umem(struct net_device *dev, struct xdp_umem *umem, u16
qid)
{
        // Validate driver-specific limitations.

        xsk_buff_dma_map(umem, dev, 0); // Check the return code!

        // Set the flag that queue qid is XSK-enabled.

        // If netdev is up, create XSK RX and TX queues.
        // Otherwise, validate the UMEM parameters, i.e. chunk_size vs MTU –
        // this is the last chance to return -EINVAL.
}
```

SETUP STAGE

```
int vnd_xsk_disable_umem(struct net_device *dev, u16 qid)
{
        struct xdp_umem *umem = xdp_get_umem_from_qid(dev, qid);

        // If netdev is up, destroy XSK RX and TX queues.
        // Clear the flag that queue qid is XSK-enabled.

        xsk_buff_dma_unmap(umem, 0);

        return 0; // Shouldn’t fail.
}
```

SETUP STAGE

```
int vnd_open_channel(...)
{
        // If an XDP program is set...
        // If this channel is flagged as XSK-enabled...
        struct xdp_umem *umem = xdp_get_umem_from_qid(dev, qid);
        // Bring up XSK queues.

        // Otherwise, configure as usual.
}
```

XSK RX QUEUE SETUP

```
// Where you call xdp_rxq_info_reg_mem_model, for XSK RX queue do instead:
err = xdp_rxq_info_reg(&rq->xdp_rxq, netdev, qid);
err = xdp_rxq_info_reg_mem_model(&rq->xdp_rxq, MEM_TYPE_XSK_BUFF_POOL, NULL);
xsk_buff_set_rxq_info(umem, &rq->xdp_rxq);
// Note: don’t forget to actually handle errors.
```

`MEM_TYPE_XSK_BUFF_POOL` must be used with XSK RX queues.

Buffer allocation will happen from the UMEM, as shown in the next slide.

XSK RX QUEUE DATA PATH

Allocate buffers with `xsk_buff_alloc` – returns a pre-filled `xdp_buff`.

Use `xsk_buff_xdp_get_frame_dma` and post a hardware descriptor.

On RX (in NAPI), use `xsk_buff_dma_sync_for_cpu` to sync DMA.

Run your XDP handling function.

Don’t unmap XSK frames.

RX XDP RESULT CODES

XDP_REDIRECT to XSKMAP: xsk_buff_free is not needed.

XDP_REDIRECT (other): xdp_convert_buff_to_frame does xsk_buff_free.

XDP_REDIRECT (errors): call xsk_buff_free.

XDP_DROP, XDP_ABORTED: call xsk_buff_free.

XDP_TX (including errors): no xsk_buff_free, it’s done by xdp_convert_buff_to_frame (however, if it fails, do xsk_buff_free).

XDP_PASS (including errors): allocate an SKB, copy data and call xsk_buff_free.

XDP_PASS EXAMPLE

```
struct sk_buff *vnd_xsk_construct_skb(struct napi_struct *napi, void *data,
u32 len)
{
        struct sk_buff *skb;

        skb = napi_alloc_skb(napi, len);
        if (unlikely(!skb)) {
                // Increase error counters.
                return NULL;
        }

        skb_put_data(skb, data, len);

        return skb;
}
```

XSK TX QUEUE DATA PATH

XSK TX queue is similar to the XDP TX queue

The application kicks TX by issuing a syscall

`.ndo_xsk_wakeup` is called to trigger an IRQ and get into NAPI poll

TX happens from NAPI poll, along with handling completions

XSK TX QUEUE DATA PATH

```
// Do TX from NAPI poll.
for (; budget; budget--) {
        struct xdp_desc desc;
        dma_addr_t dma;

        if (!xsk_umem_consume_tx(umem, &desc))
                break;
        dma = xsk_buff_raw_get_dma(umem, desc.addr);
        xsk_buff_raw_dma_sync_for_device(umem, dma, desc.len);
        // Transmit the packet (dma, desc.len).
        flush = true;
}

if (flush) {
        // Ring the doorbell.
        xsk_umem_consume_tx_done(umem);
}
```

XSK TX QUEUE DATA PATH

```
// Poll completions from hardware as usual.

// Count them:
xsk_frames++;

// Call this in the end:
if (xsk_frames)
        xsk_umem_complete_tx(umem, xsk_frames);
```

EXTRA STUFF

UNALIGNED CHUNKS

Extension of AF_XDP to support frames that start at unaligned addresses

Transparent to the compatible drivers

The driver can check the `unaligned` field of `struct xsk_buff_pool` or the `XDP_UMEM_UNALIGNED_CHUNK_FLAG` flag of `struct xdp_umem`

NEED_WAKEUP

A performance feature to avoid unnecessary busy polling

On RX: avoid in-driver polling when the fill ring is empty

On TX: avoid syscall flood when the driver is going to wake up anyway

NEED_WAKEUP ON RX

```
// Call this function after posting hardware descriptors
// alloc_err is true if xsk_buff_alloc returned NULL
bool vnd_xsk_update_rx_wakeup(struct xsk_buff_pool *pool, bool alloc_err)
{
        if (!xsk_uses_need_wakeup(pool))
                return alloc_err; // return true means reschedule NAPI

        if (alloc_err)
                xsk_set_rx_need_wakeup(pool);
        else
                xsk_clear_rx_need_wakeup(pool);

        return false;
}
```

NEED_WAKEUP ON TX

```
void vnd_xsk_update_tx_wakeup(struct xsk_buff_pool *pool)
{
        if (!xsk_uses_need_wakeup(pool))
                return;

        if (/* XSK TX queue is empty */)
                xsk_set_tx_need_wakeup(pool);
        else
                xsk_clear_tx_need_wakeup(pool);
}

// Call it from NAPI twice to avoid a race condition:
vnd_poll_xsk_completions(queue);
vnd_xsk_update_tx_wakeup(pool);
vnd_xsk_tx(pool, VND_TX_XSK_POLL_BUDGET);
vnd_xsk_update_tx_wakeup(pool);
```

NEED_WAKEUP RACE CONDITION FIX FOR TX

Race condition

Driver	Application
Transmit packets	
	Put new packets to transmit
	Query need_wakeup – it’s false
Hardware queue is empty, set need_wakeup	
Waits for the wakeup syscall	Doesn’t call the wakeup syscall

Fix

```
vnd_poll_xsk_completions(queue);
vnd_xsk_update_tx_wakeup(pool); // Can become true
vnd_xsk_tx(pool, VND_TX_XSK_POLL_BUDGET);
vnd_xsk_update_tx_wakeup(pool); // Can become false
```

QUEUE ALLOCATION SCHEME

i40e way

XSK on channel X replaces normal queues

RSS is broken by default

mlx5 way

Channels 0..N-1 are for normal traffic

Queue IDs N..2*N-1 are for XSK

Fallback to copy mode is broken