Bringing TSO/GRO and Jumbo frames to XDP

Lorenzo Bianconi
Eelco Chaudron
Jesper Dangaard Brouer
Toke Høiland-Jørgensen

Linux Plumbers – Networking & BPF Summit
September 2021

XDP technical requirements

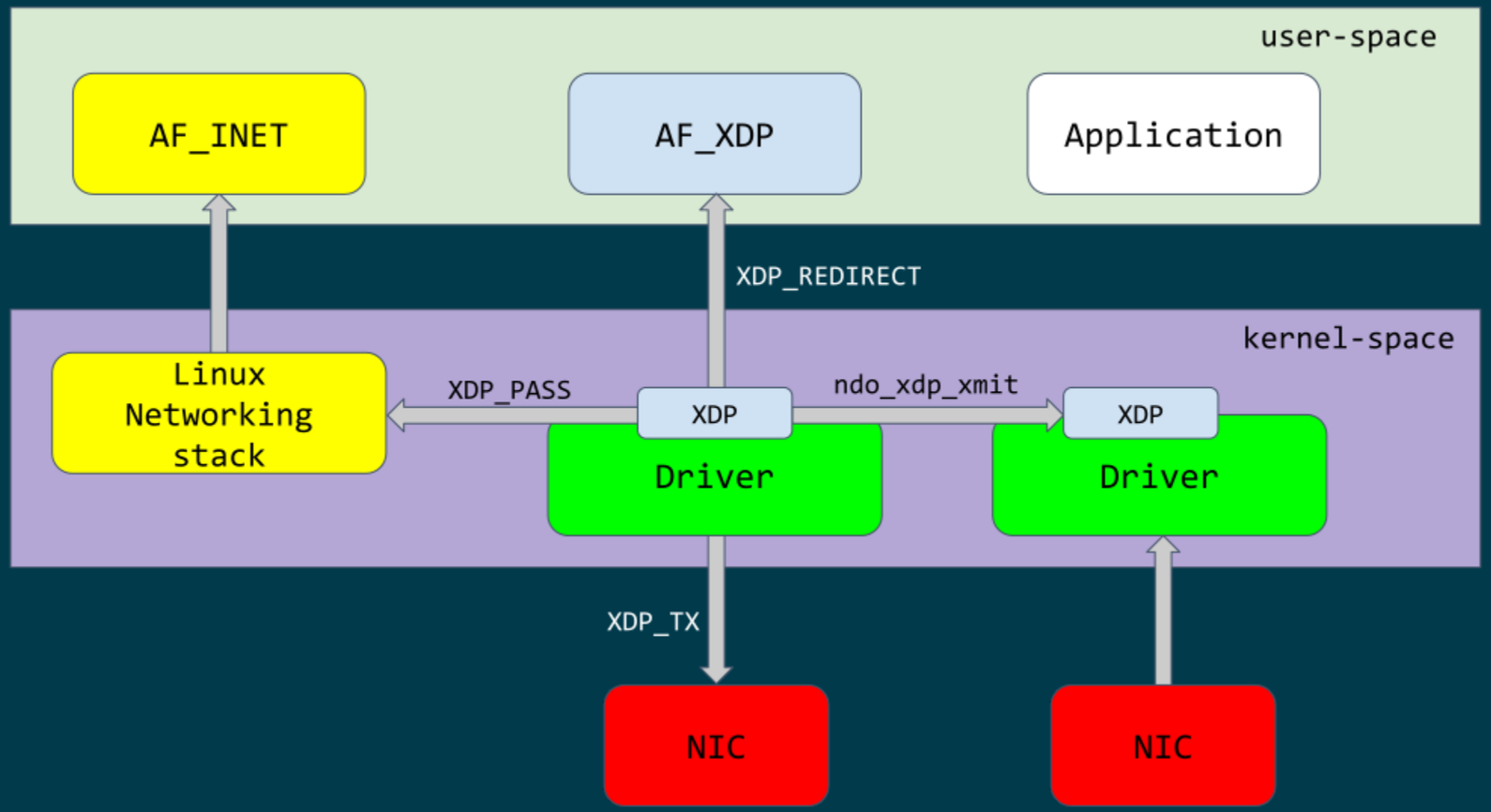
- Quick introduction to XDP
- XDP technical requirements
- XDP multi-buffers use-cases/requirements

What is XDP?

XDP (eXpress Data Path) is a Linux in-kernel fast-path

- New programmable layer in-front of traditional network stack
  - Read, modify, drop, redirect or pass
  - For L2-L3 use-cases: seeing x10 performance improvements!
- Avoiding memory allocations
  - No SKB allocations and no-init (memset zero 4 cache-lines)
- Adaptive bulk processing of frames
- Very early access to frame (in driver code after DMA sync)
- Ability to skip (large parts) of kernel code
  - Evolve XDP via BPF-helpers

XDP architecture



XDP current requirements

- XDP frame in physical contiguous memory
  - BPF Direct-Access for validating correctness
  - No paged frames support, data cannot be split across pages
  - Read and Write access to the DMA buffers
  - Disable jumbo frames (packet < PAGE_SIZE) loading a BPF program
- XDP headroom for xdp_frame area
  - add push/pop header through bpf_xdp_adjust_head()
- Reserve tailroom for skb_shared_info and rely on build_skb() on XDP_PASS
- Cannot allocate page fragments to support it (e.g. through napi_alloc_skb())
- Rx buffers must be recycled to get high speed!

Multi-buffers support for XDP

- XDP multi-buffers use cases:
  - Enable Jumbo frames (larger than 3502 MTU settings)
  - GRO/TSO for XDP_REDIRECT
  - Packet header split
  - Handling GRO SKBs in veth/cpumap/generic-XDP
- Constraints:
  - Fast conversion from xdp_buff/xdp_frame to SKBs
  - Support non-linear buffer and not slow down single buffer use case
  - How to satisfy BPF Direct-Access (DA) design?

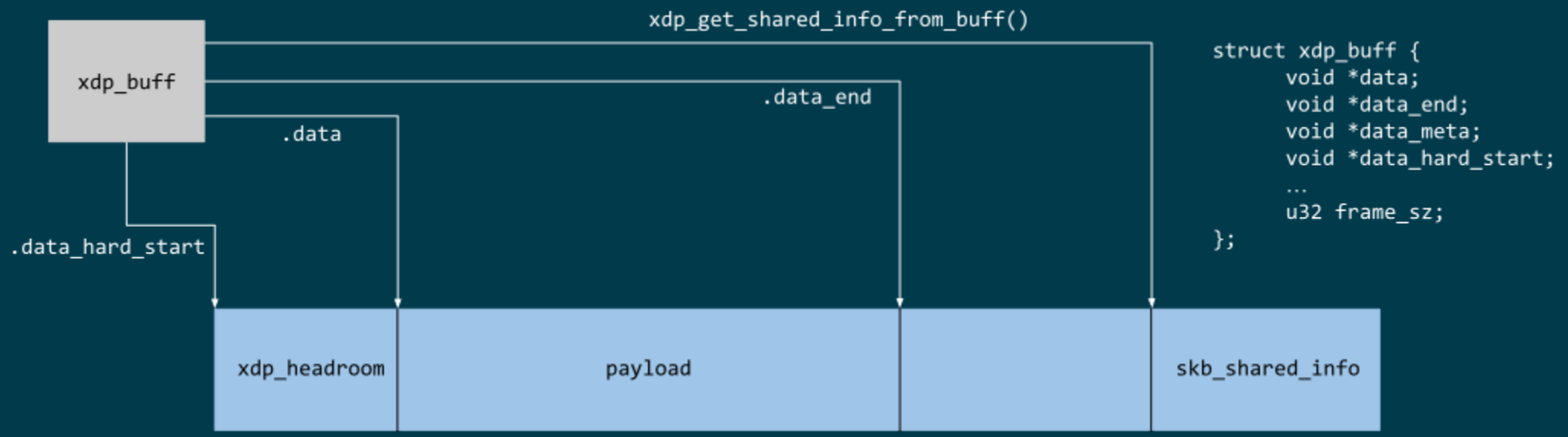
XDP multi-buffers

Work-in-progress upstream patchset V14

- xdp_buff and xdp_frame
- General requirements/proposed solution
- xdp_adjust_data BPF helper
- XDP multi-buffers for mvneta

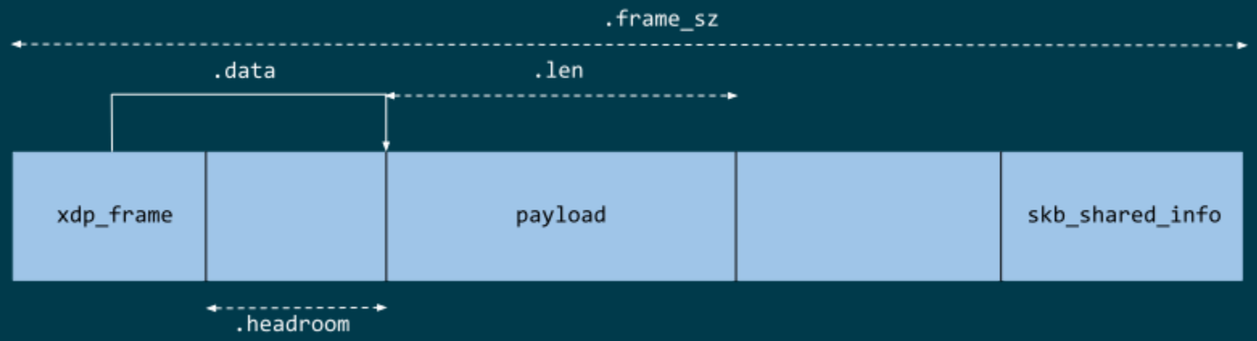
xdp_buff: let's not use the skb (1/2)

- xdp_buff: metadata container for received frames
  - valid only in the driver NAPI context
  - skb_shared_info to rely on build_skb() for XDP_PASS



xdp_frame: let's not use the skb (2/2)

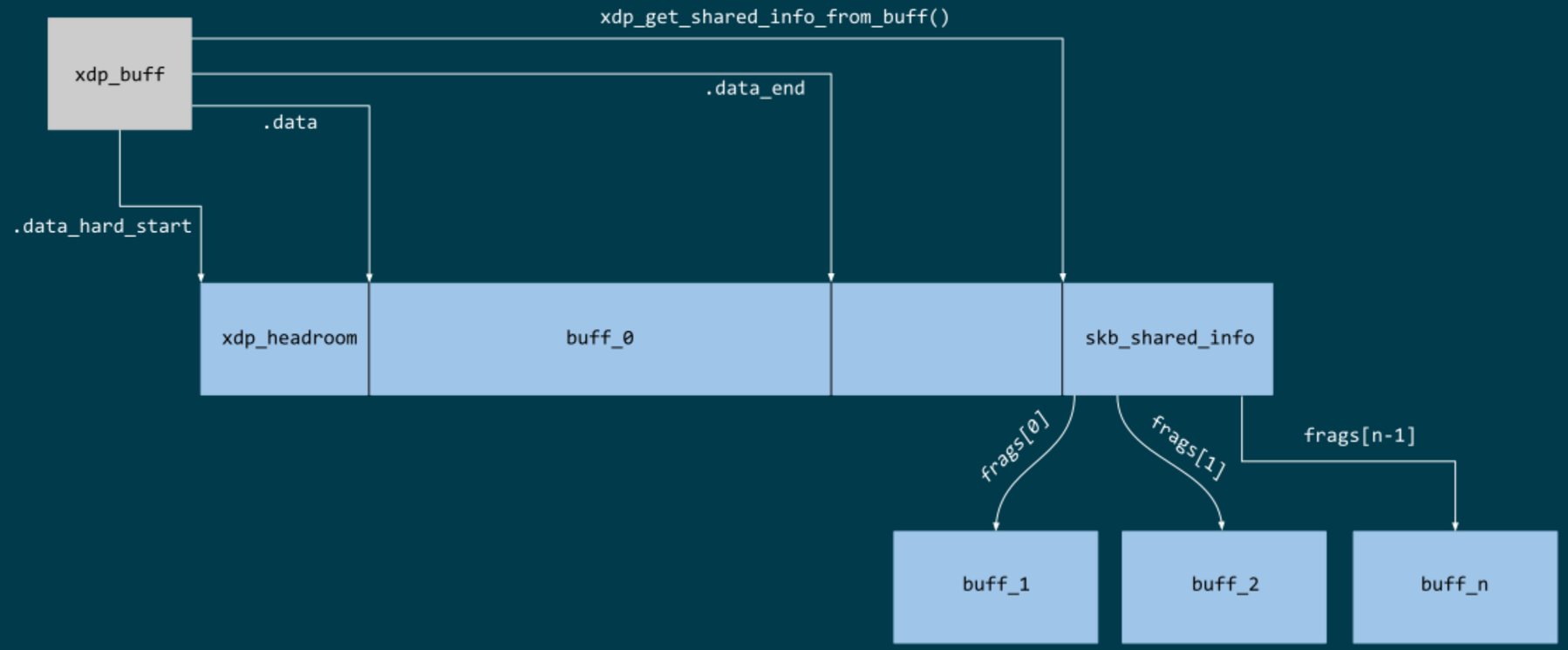
- xdp_frame: metadata container valid outside of the NAPI context
  - XDP_REDIRCT
  - no memory allocation in the hot path
    - stored in the buffer headroom - memory constraints
    - must fit in a single cache-line




```
struct xdp_frame {
        void *data;
        u16 len;
        u16 headroom;
        u32 meta_size:8;
        u32 frame_sz:24;
        ...
};
```

XDP multi-buffers architecture

- Use same layout as skb_shared_info allows faster SKB conversion



Keep single buffer fast!

- Add flags capability field in xdp_buff/xdp_frame
  - XDP_FLAGS_MULTI_BUFF bit

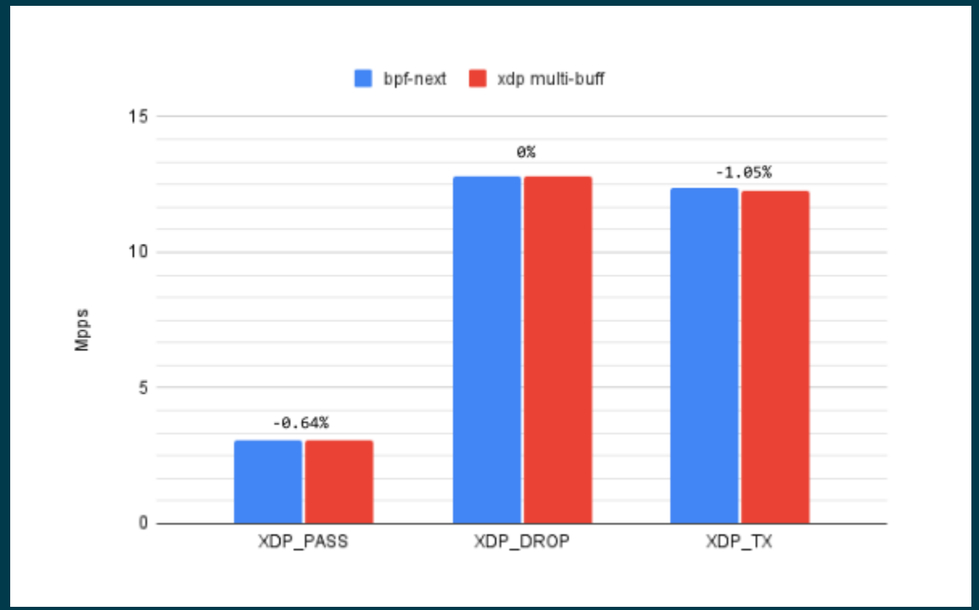
```
struct xdp_buff {
        ...
        u32 flags; /* supported values defined in xdp_buff_flags */
        ...
};

enum xdp_buff_flags {
        XDP_FLAGS_MULTI_BUFF                = BIT(0), /* non-linear xdp buff */
        ...
};
```

- Driver will set XDP_FLAGS_MULTI_BUFF only for multi-descriptor frames
- XDP code will check fragments only if XDP_FLAGS_MULTI_BUFF is set
  - Point: Avoid touching cache-line (for skb_shared_info) unless needed

XDP multi-buffers overhead

- Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz
  - 82599ES 10-Gigabit SFI/SFP+ (ixgbe)
- pktgen_sample03_burst_single_flow.sh (64B packet size)



XDP multi-buffers: stack support

- XDP return path

```
void xdp_return_frame_rx_napi(struct xdp_frame *xdpf)
{
        struct skb_shared_info *sinfo = xdp_get_shared_info_from_frame(xdpf);
        int i;

        if (likely(!xdp_frame_is_mb(xdpf)))
                goto out;

        for (i = 0; i < sinfo->nr_frags; i++) {
                struct page *page = skb_frag_page(&sinfo->frags[i]);

                __xdp_return(page_address(page), &xdpf->mem, true, NULL);
        }
out:
        __xdp_return(xdpf->data, &xdpf->mem, true, NULL);
}
```

XDP multi-buffers patchset - BPF-helpers

New BPF-helpers

XDP multi-buffers: new BPF helpers (1/5)

- bpf_xdp_get_buff_len:
  - BPF helper to compute non-linear buffer length
    - linear + paged

```
BPF_CALL_1(bpf_xdp_get_buff_len, struct  xdp_buff*, xdp)
{
      u64 len = xdp->data_end - xdp->data; /* linear length */

      if (unlikely(xdp_buff_is_mb(xdp))) {
            struct skb_shared_info *sinfo;

            sinfo = xdp_get_shared_info_from_buff(xdp);
            len += sinfo->xdp_frags_size; /* paged length */
      }

      return len;
}
```

XDP multi-buffers: new BPF helpers (2/5)

- bpf_xdp_mb_adjust_tail:
  - run by bpf_xdp_adjust_tail() helper for mb xdp_buff

```
static int bpf_xdp_mb_adjust_tail(struct xdp_buff *xdp, int offset)
{
      struct skb_shared_info *sinfo = xdp_get_shared_info_from_buff(xdp);
      if (offset >= 0) { /* increase last frag */
            ...
            skb_frag_size_set(frag, skb_frag_size(&sinfo->frags[sinfo->nr_frags - 1]) + offset);
      } else { /* shrink frags */
            for (i = sinfo->nr_frags - 1; i >= 0 && abs(offset) > 0; i--) {
                  if (unlikely(skb_frag_size(&sinfo->frags[i]) == shrink)) {
                        __xdp_return(page_address(skb_frag_page(&sinfo->frags[i])), ...);
                  } else {
                        skb_frag_size_set(&sinfo->frags[i],
                                          skb_frag_size(&sinfo->frags[i]) - shrink);
                        break;
                  }
            }
      }
}
```

XDP multi-buffers: new BPF helpers (3/5)

- bpf_xdp_adjust_data:

```
BPF_CALL_2(bpf_xdp_adjust_data, struct xdp_buff *, xdp, u32, offset)
{
      struct skb_shared_info *sinfo = xdp_get_shared_info_from_buff(xdp);
      u32 base_offset = xdp->mb.headlen; /* xdp->mb.headlen is linear length */
      ...
      if (offset < xdp->mb.headlen) { /* linear area */
            xdp->data = xdp->data_hard_start + xdp->mb.headroom + offset;
            xdp->data_end = xdp->data_hard_start + xdp->mb.headroom + xdp->mb.headlen;
            return 0;
      }
      for (i = 0; i < sinfo->nr_frags; i++) { /* paged area */
            if (offset < base_offset + skb_frag_size(&sinfo->frags[i])) {
                  xdp->data = skb_frag_address(&sinfo->frags[i]) + offset - base_offset;
                  xdp->data_end = skb_frag_address(&sinfo->frags[i]) +
                                  skb_frag_size(&sinfo->frags[i]);
                  break;
            }
            base_offset += skb_frag_size(&sinfo->frags[i]);
      }
}
```

XDP multi-buffers: new BPF helpers (4/5)

- bpf_xdp_adjust_data() move data pointers in the selected fragment
  - data and data_end must be moved back to the original position

```
int _xdp_adjust_frags(struct xdp_md *xdp)
{
        __u8 *data_end = (void *)(long)xdp->data_end;
        __u8 *data = (void *)(long)xdp->data;
        ...
        int base_offset = bpf_xdp_adjust_data(xdp, 5000);
        ...
        data_end = (void *)(long)xdp->data_end; /* must be reloaded */
        data = (void *)(long)xdp->data;
        ...
        if (*data != 0xaa) /* marker */
                goto out;
        *data = 0xbb; /* update the marker */
out:
        bpf_xdp_adjust_data(xdp, 0);
        return XDP_PASS;
}
```

XDP multi-buffers: new BPF helpers (5/5)

- bpf_xdp_output and bpf_perf_event_output:
  - Helpers are updated and can copy the content of all buffers

```
static unsigned long bpf_xdp_copy(void *dst_buff, const void *ctx,
                                  unsigned long off, unsigned long len)
{
        if (likely(!xdp_buff_is_mb(xdp))) { /* single buffer */
                memcpy(dst_buff, xdp->data + off, len);
                return 0;
        }
        ...
        if (off < base_len) { /* copy data from the base buffer */
                ...
                memcpy(dst_buff, xdp->data + off, copy_len);
        }
        sinfo = xdp_get_shared_info_from_buff(xdp);
        for (i = 0; len && i < sinfo->nr_frags; i++) { /* copy frag remaining data */
                ...
                memcpy(dst_buff, skb_frag_address(&sinfo->frags[i]) + frag_off, copy_len);
        }
}
```

XDP multi-buffers patchset - driver changes

Changes for Marvell driver: mvneta

XDP multi-buffers support for mvneta (1/5)

- Modify drivers rx NAPI loop
  - Process all RX descriptor segments building xdp_buff
    - `mvneta_swbm_rx_frame()`
    - `mvneta_swbm_add_rx_fragment()`
      - set XDP_FLAGS_MULTI_BUFF for multi-descriptor frames
  - Run the BPF program when all descriptors are processed
  - Change XDP_TX and ndo_xdp_xmit to map non-linear buffers
    - `mvneta_xdp_submit_frame()`
  - Remove MTU check loading the BPF program
    - `mvneta_xdp_setup()`

XDP multi-buffers support for mvneta (2/5)

- `mvneta_swbm_add_rx_fragment()` runs on nth descriptor (n > 1)

```
void mvneta_swbm_add_rx_fragment(struct xdp_buff *xdp, ...)
{
      struct skb_shared_info *sinfo = xdp_get_shared_info_from_buff(xdp);
      ...
      if (data_len > 0 && sinfo->nr_frags < MAX_SKB_FRAGS) {
            skb_frag_t *frag = &sinfo->frags[sinfo->nr_frags++];
            skb_frag_off_set(frag, offset);
            skb_frag_size_set(frag, data_len);
            __skb_frag_set_page(frag, page);

            if (!xdp_buff_is_mb(xdp)) {
                  sinfo->xdp_frags_size = *size; /* non-linear size */
                  xdp_buff_set_mb(xdp); /* set XDP_FLAGS_MULTI_BUFF */
            }
      }
      ...
      sinfo->xdp_frags_truesize = sinfo->nr_frags * PAGE_SIZE; /* non-linear truesize */
      ...
}
```

XDP multi-buffers support for mvneta (3/5)

```
struct sk_buff *mvneta_swbm_build_skb(struct xdp_buff *xdp, ..)
{
      struct skb_shared_info *sinfo = xdp_get_shared_info_from_buff(xdp);
      ...
      skb = build_skb(xdp->data_hard_start, PAGE_SIZE);
      ...
      if (unlikely(xdp_buff_is_mb(xdp)))
            xdp_update_skb_shared_info(skb, sinfo->nr_frags, sinfo->xdp_frags_size,
                                       sinfo->xdp_frags_truesize, ...);
      ...
}

static inline void
xdp_update_skb_shared_info(struct sk_buff *skb, u8 nr_frags, unsigned int size,
                           unsigned int truesize, bool pfmemalloc)
{
      skb_shinfo(skb)->nr_frags = nr_frags;
      skb->len += size;
      skb->data_len += size;
      skb->truesize += truesize;
      skb->pfmemalloc |= pfmemalloc;
}
```

XDP multi-buffers support for mvneta (4/5)

```
static int mvneta_xdp_submit_frame(..., struct xdp_frame *xdpf, ...)
{
        struct skb_shared_info *sinfo = xdp_get_shared_info_from_frame(xdpf);
        ...
        for (i = 0; i < sinfo->nr_frags + 1; i++) {
                skb_frag_t *frag  = i ? &sinfo->frags[i - 1] : NULL;
                int len = frag ? skb_frag_size(frag) : xdpf->len;
                if (dma_map) { /* ndo_xdp_xmit */
                        void *data = unlikely(frag) ? skb_frag_address(frag) : xdpf->data;
                        dma_addr = dma_map_single(dev, data, len, DMA_TO_DEVICE);
                } else { /* XDP_TX */
                        struct page *page = unlikely(frag) ? skb_frag_page(frag)
                                                           : virt_to_page(xdpf->data);
                        dma_addr = page_pool_get_dma_addr(page);
                        dma_sync_single_for_device(dev, dma_addr, len, DMA_BIDIRECTIONAL);
                }
                ...
                tx_desc->buf_phys_addr = dma_addr;
                tx_desc->data_size = len;
                ...
        }
}
```

XDP multi-buffers support for mvneta (5/5)

- we can now remove MTU constraints in mvneta_xdp_setup (.ndo_bpf) to support Jumbo frames and GRO/TSO for XDP_REDIRECT

```
root@espresso-bin:~# ip link show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 xdp qdisc mq state UP mode DEFAULT group default qlen 1024
    link/ether f0:ad:4e:09:6b:57 brd ff:ff:ff:ff:ff:ff
    prog/xdp id 11 tag 3b185187f1855c4c jited
```

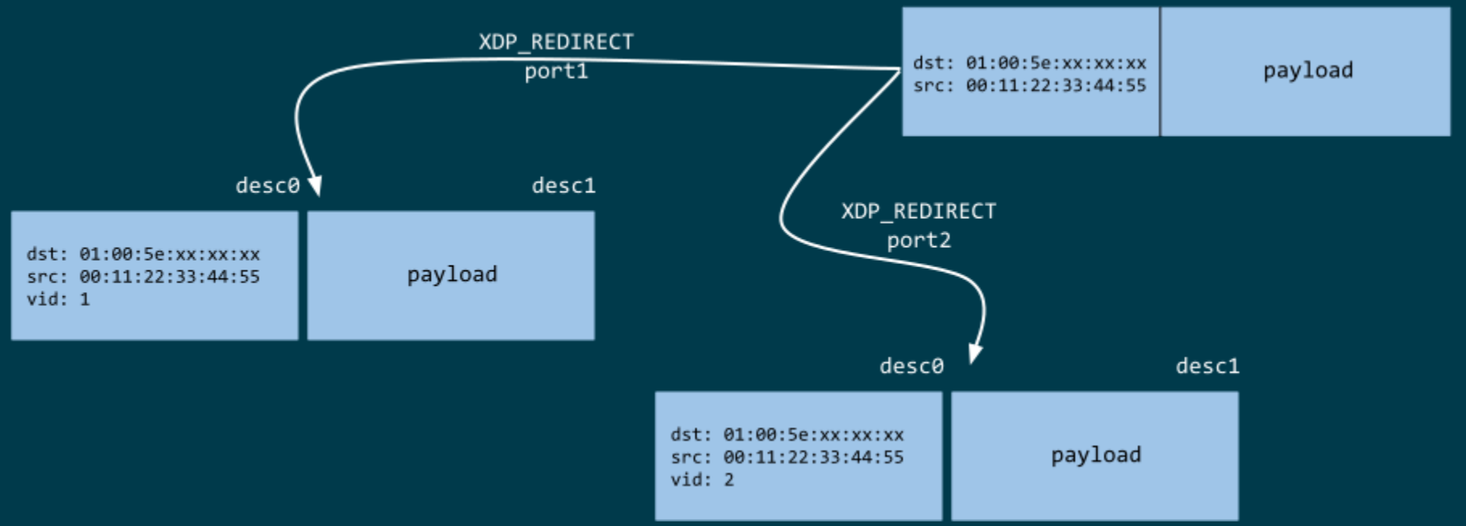
Future development

XDP multi-buffers: future development (1/2)

- XDP driver capabilities
  - XDP multi-buffers for XDP_REDIRECT
- driver support
  - intel i40e (work-in-progress)
  - ena (work-in-progress)
  - virtio-net
  - ...

XDP multi-buffers: future development (2/2)

- Relying on hw Scatter-Gather (SG) support to modify shared buffers w/o allocating memory
  - reserve buffers to push/pop headers (e.g VLAN tag)
  - XDP multicast



Q&A:

- https://github.com/xdp-project
- https://xdp-project.net