XDP meta-data Acceleration

Jan 2020

Saeed Mahameed

Intro + Motivation

● *XDP BFP programs are only allowed to see and touch packet data

● XDP can’t properly work with some offloads – Vlan striping – Csum complete

● HW offloads and hints are great for acceleration

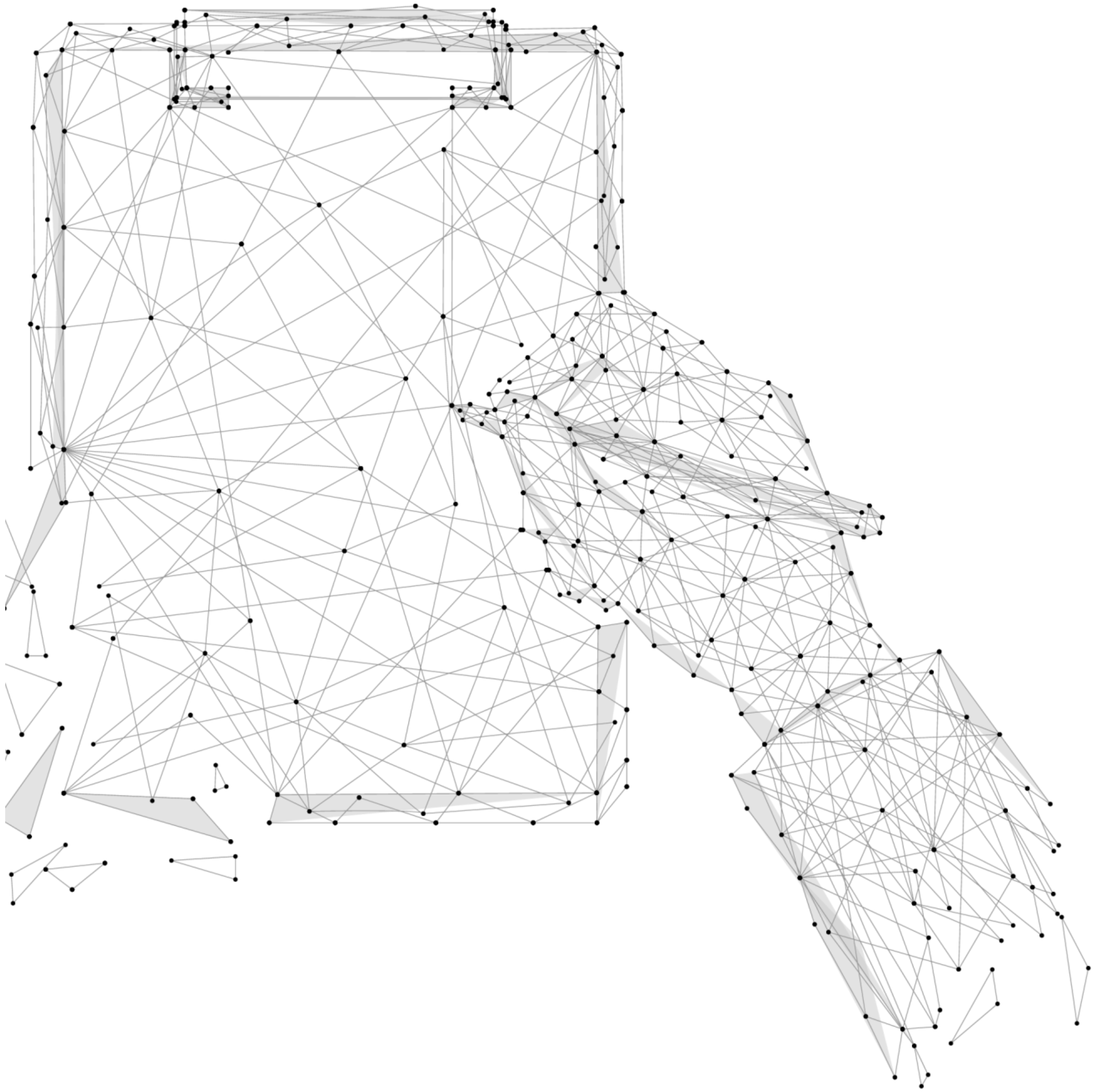
● Low hanging fruit: Legacy and new NICs, offer a wide variety of hints, offloads and parsing capabilities.

● Waiting for a reliable and fast full BPF offload support will take years..

Agenda

XDP meta- data

- Current state + API
- The easy part
- Requirements
- BTF – BPF Type format
- High level design
- Driver BTF registration
- User API
- Example user program
- Kernel Examples + Compilation guide



XDP->data_meta

- Most drivers preserve 128bytes for headroom per XDP packet/page, that is untouched/seen by hardware

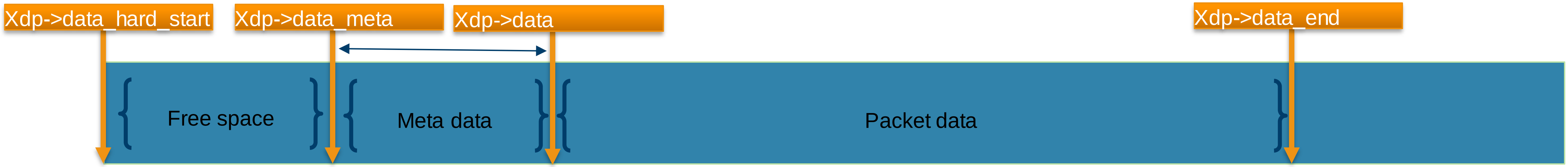
- This headroom is accessible by XDP BPF programs via the following API:

```
struct xdp_buff *xdp {
        ...
        void *data_meta;
       ...
}
```

- xdp_set_data_meta_invalid(&xdp);

Default, XDP data meta is not used by driver, set by drivers who don't support meta data.

- XDP->data_meta, occupy the free headroom buffer just before the packet data (xdp->data);



bpf_xdp_adjust_meta

- **int bpf_xdp_adjust_meta(struct xdp_buff *xdp_md, int delta)**

Description

Adjust the address pointed by *xdp_md*->**data_meta** by
delta (which can be positive or negative). Note that
this operation modifies the address stored in
xdp_md->**data**, so the latter must be loaded only after
the helper has been called.

The use of *xdp_md*->**data_meta** is optional and programs
are not required to use it. The rationale is that when
the packet is processed with XDP (e.g. as DoS filter),
it is possible to push further meta data along with it
before passing to the stack, a [snip …]
A call to this helper is susceptible to change the
underlying packet buffer. Therefore, at load time, all
checks on pointers previously done by the verifier are
invalidated and must be performed again, if the helper
is used in combination with direct packet access.

bpf_xdp_adjust_head

- **int bpf_xdp_adjust_head(struct xdp_buff ****xdp_md*, **int** *delta***)**

  Description

  Adjust (move) *xdp_md***->data** by *delta* bytes. Note that it is possible to use a negative value for *delta*. This helper can be used to prepare the packet for pushing or popping headers.

  A call to this helper is susceptible to change the underlying packet buffer. Therefore, at load time, all checks on pointers previously done by the verifier are invalidated and must be performed again, if the helper is used in combination with direct packet access.

  Return 0 on success, or a negative error in case of failure.

- * Caution: if used when xdp->data_meta is valid, it will memove xdp->data_meta further back.

The easy part

- driver on XDP RX packet :

```
 xdp_buff.data_meta = xdp_buff->data - sizeof(meta_data);
*xdp_buff.data_meta = meta_data;
```

- XDP user program:

```
meta_data = (struct meta_data*)xdp_buff->data_meta;
```

- XDP->data_meta is already used by some XDP program to pass information to the stack
  - pass prio/socket/SKB Mark info from XDP prog to TC layer
  - Pass queue info to the stack

- So why do we need a new design for passing meta data from driver to xdp progs ?
  - What is the format of the meta data passed from different drivers?
  - We can't just use a fixed well-known type (defeats the whole dynamic purpose of XDP).
  - keep programs as generic as possible, run on any NIC, with or without meta data offloads.
  - Future hw support, no driver copy

Requirements

- Generic meta data types: use BTF
  Generic data type exchange between driver and XDP programs

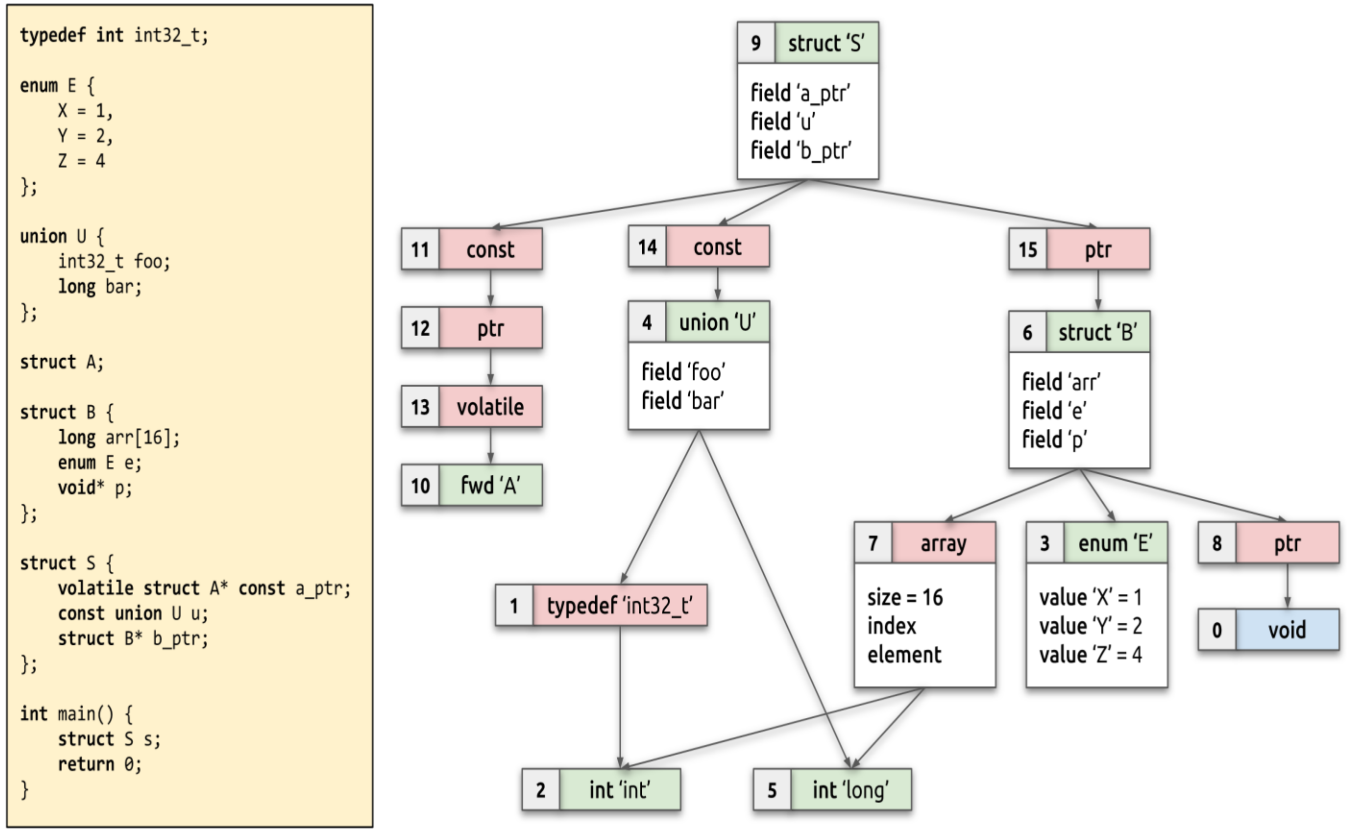
- Dynamic meta data settings:
  in order to work with the minimal set of required meta data(s) in a XDP program

- XDP meta data direct access:
  either at run-time or compile time, no lookups to find the needed meta data.

BTF (BPF Type Format)

- BTF (BPF Type Format) is the metadata format which encodes the debug info related to BPF program/map. The name BTF was used initially to describe data types. The BTF was later extended to include function info for defined subroutines, and line info for source/line information.

- https://www.kernel.org/doc/html/latest/bpf/btf.html



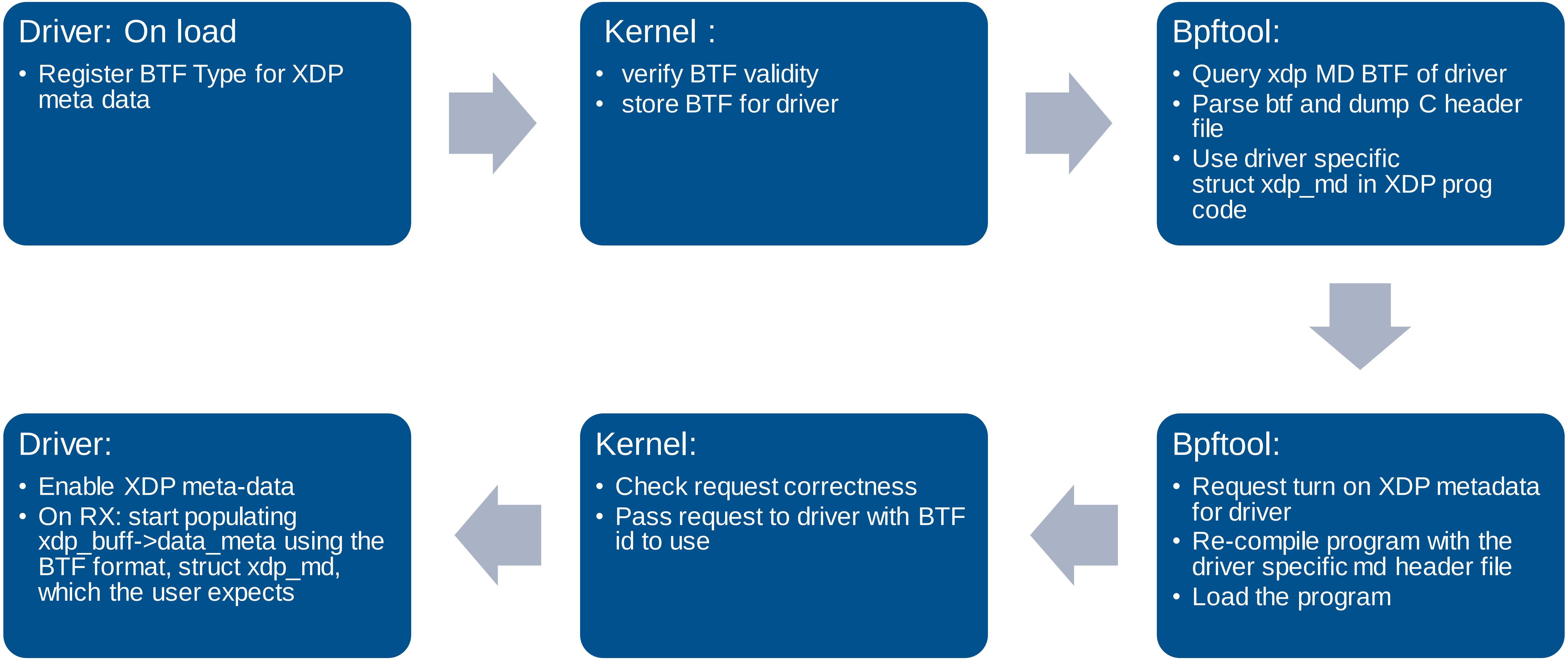
```
typedef int int32_t;

enum E {
    X = 1,
    Y = 2,
    Z = 4
};

union U {
    int32_t foo;
    long bar;
};

struct A;

struct B {
    long arr[16];
    enum E e;
    void* p;
};

struct S {
    volatile struct A* const a_ptr;
    const union U u;
    struct B* b_ptr;
};

int main() {
    struct S s;
    return 0;
}
```

High level design

- Driver+firmware keep layout of the metadata in BTF format
- User to query the driver and generate normal C header file based on BTF in the given NIC
- During sys_bpf(prog_load) the kernel checks (via supplied BTF)
- Every NIC can have their own layout of metadata and its own meaning of the fields
- Standardize at least a few common fields like hash

- proposed by Alexei Starovoitov and Daniel Borkmann ML discussion: https://www.spinics.net/lists/netdev/msg509820.html

HLD: Data flow



Driver: On load

- Register BTF Type for XDP meta data

Kernel :

- verify BTF validity
- store BTF for driver

Bpftool:

- Query xdp MD BTF of driver
- Parse btf and dump C header file
- Use driver specific struct xdp_md in XDP prog code

Bpftool:

- Request turn on XDP metadata for driver
- Re-compile program with the driver specific md header file
- Load the program

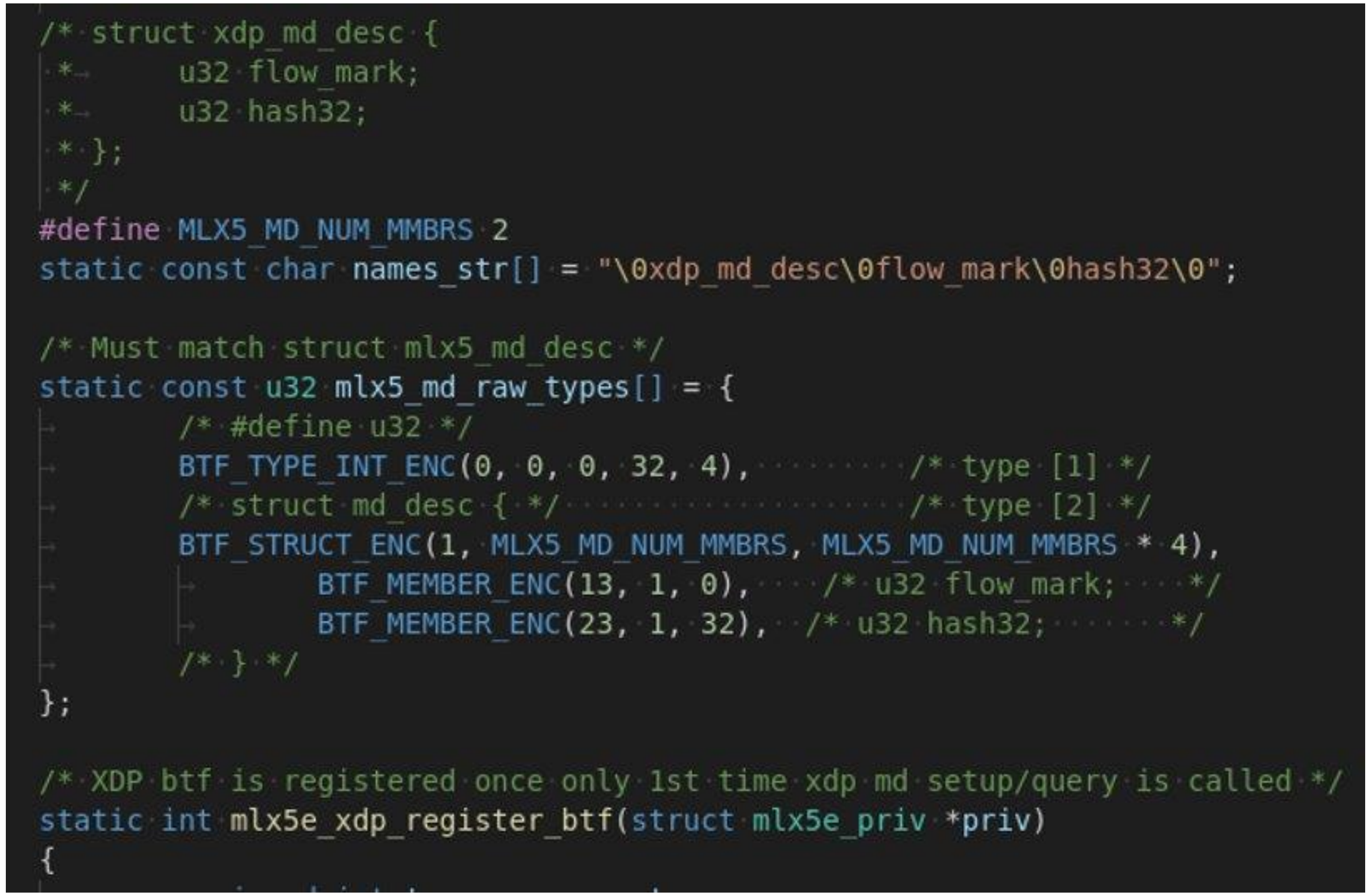
Kernel:

- Check request correctness
- Pass request to driver with BTF id to use

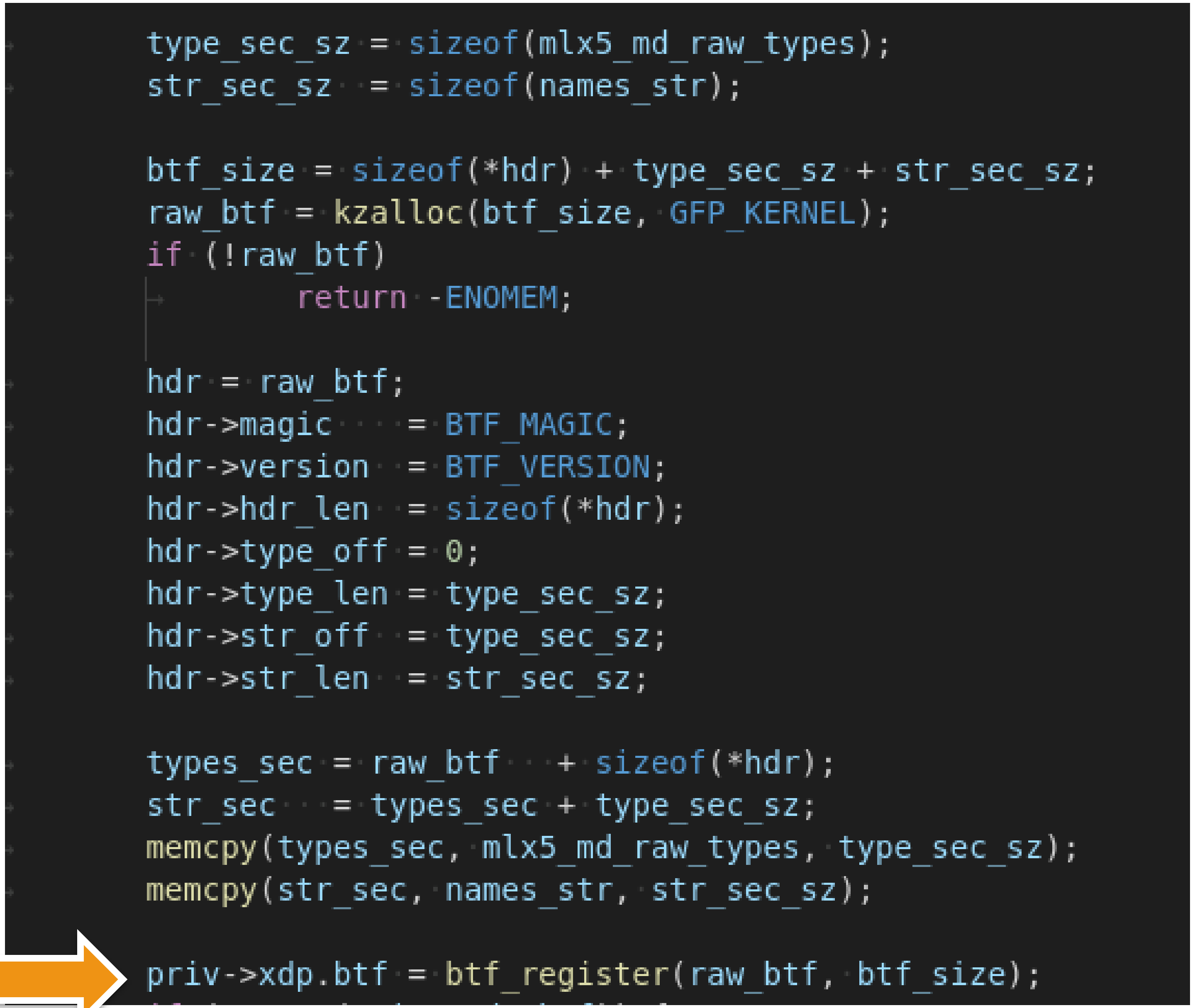
Driver:

- Enable XDP meta-data
- On RX: start populating xdp_buff->data_meta using the BTF format, struct xdp_md, which the user expects

Driver BTF registration



```
/* struct xdp_md_desc {
 *      u32 flow_mark;
 *      u32 hash32;
 * };
 */
#define MLX5_MD_NUM_MMBRS 2
static const char names_str[] = "\0xdp_md_desc\0flow_mark\0hash32\0";

/* Must match struct mlx5_md_desc */
static const u32 mlx5_md_raw_types[] = {
        /* #define u32 */
        BTF_TYPE_INT_ENC(0, 0, 0, 32, 4),          /* type [1] */
        /* struct md_desc { */                     /* type [2] */
        BTF_STRUCT_ENC(1, MLX5_MD_NUM_MMBRS, MLX5_MD_NUM_MMBRS * 4),
                BTF_MEMBER_ENC(13, 1, 0),     /* u32 flow_mark;    */
                BTF_MEMBER_ENC(23, 1, 32),   /* u32 hash32;       */
        /* } */
};

/* XDP btf is registered once only 1st time xdp md setup/query is called */
static int mlx5e_xdp_register_btf(struct mlx5e_priv *priv)
{
```



```
        type_sec_sz = sizeof(mlx5_md_raw_types);
        str_sec_sz  = sizeof(names_str);

        btf_size = sizeof(*hdr) + type_sec_sz + str_sec_sz;
        raw_btf = kzalloc(btf_size, GFP_KERNEL);
        if (!raw_btf)
                return -ENOMEM;

        hdr = raw_btf;
        hdr->magic    = BTF_MAGIC;
        hdr->version  = BTF_VERSION;
        hdr->hdr_len  = sizeof(*hdr);
        hdr->type_off = 0;
        hdr->type_len = type_sec_sz;
        hdr->str_off  = type_sec_sz;
        hdr->str_len  = str_sec_sz;

        types_sec = raw_btf   + sizeof(*hdr);
        str_sec   = types_sec + type_sec_sz;
        memcpy(types_sec, mlx5_md_raw_types, type_sec_sz);
        memcpy(str_sec, names_str, str_sec_sz);

        priv->xdp.btf = btf_register(raw_btf, btf_size);
```

Netlink User API

- $ /usr/local/sbin/bpftool net xdp help

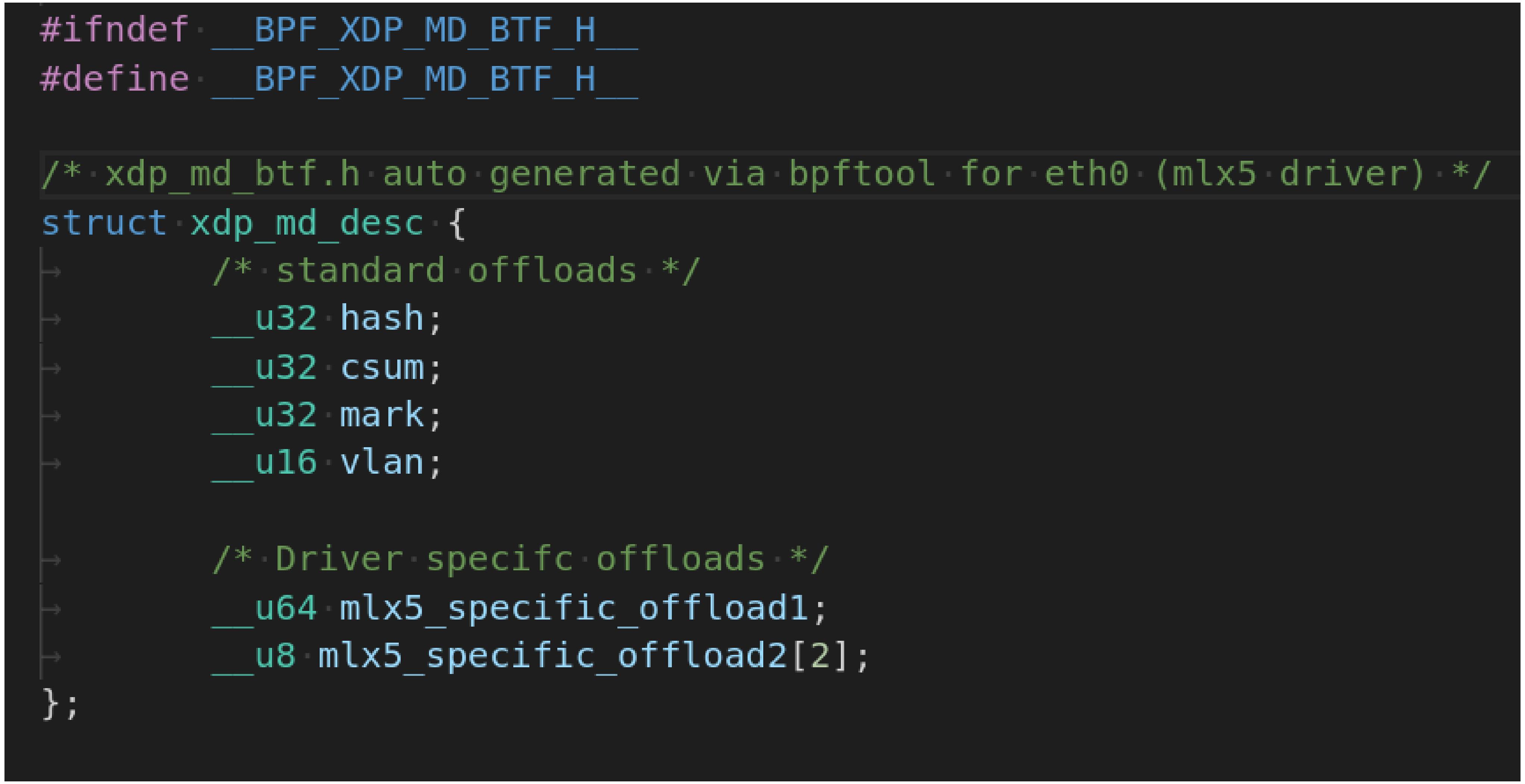
```
Usage: /usr/local/sbin/bpftool xdp xdp { show | list | set | md_btf} [dev <devname>]
       /usr/local/sbin/bpftool xdp help
```

- Query xdp metdata support

```
$ /usr/local/sbin/bpftool net xdp
xdp:
mlx0(3) md_btf_id(1) md_btf_enabled(0)
```

Query and Dump driver BTF meta data type

- /usr/local/sbin/bpftool net xdp md_btf cstyle dev mlx0



```c
#ifndef __BPF_XDP_MD_BTF_H__
#define __BPF_XDP_MD_BTF_H__

/* xdp_md_btf.h auto generated via bpftool for eth0 (mlx5 driver) */
struct xdp_md_desc {
	/* standard offloads */
	__u32 hash;
	__u32 csum;
	__u32 mark;
	__u16 vlan;

	/* Driver specifc offloads */
	__u64 mlx5_specific_offload1;
	__u8 mlx5_specific_offload2[2];
};
```

Driver XDP metadata BTF state on/off

- enable meta data on mlx0

```
$ /usr/local/sbin/bpftool net xdp set dev mlx0 md_btf on
```

- verify enabled ?

```
$ /usr/local/sbin/bpftool net xdp

xdp:

mlx0(3) md_btf_id(1) md_btf_enabled(1)
```

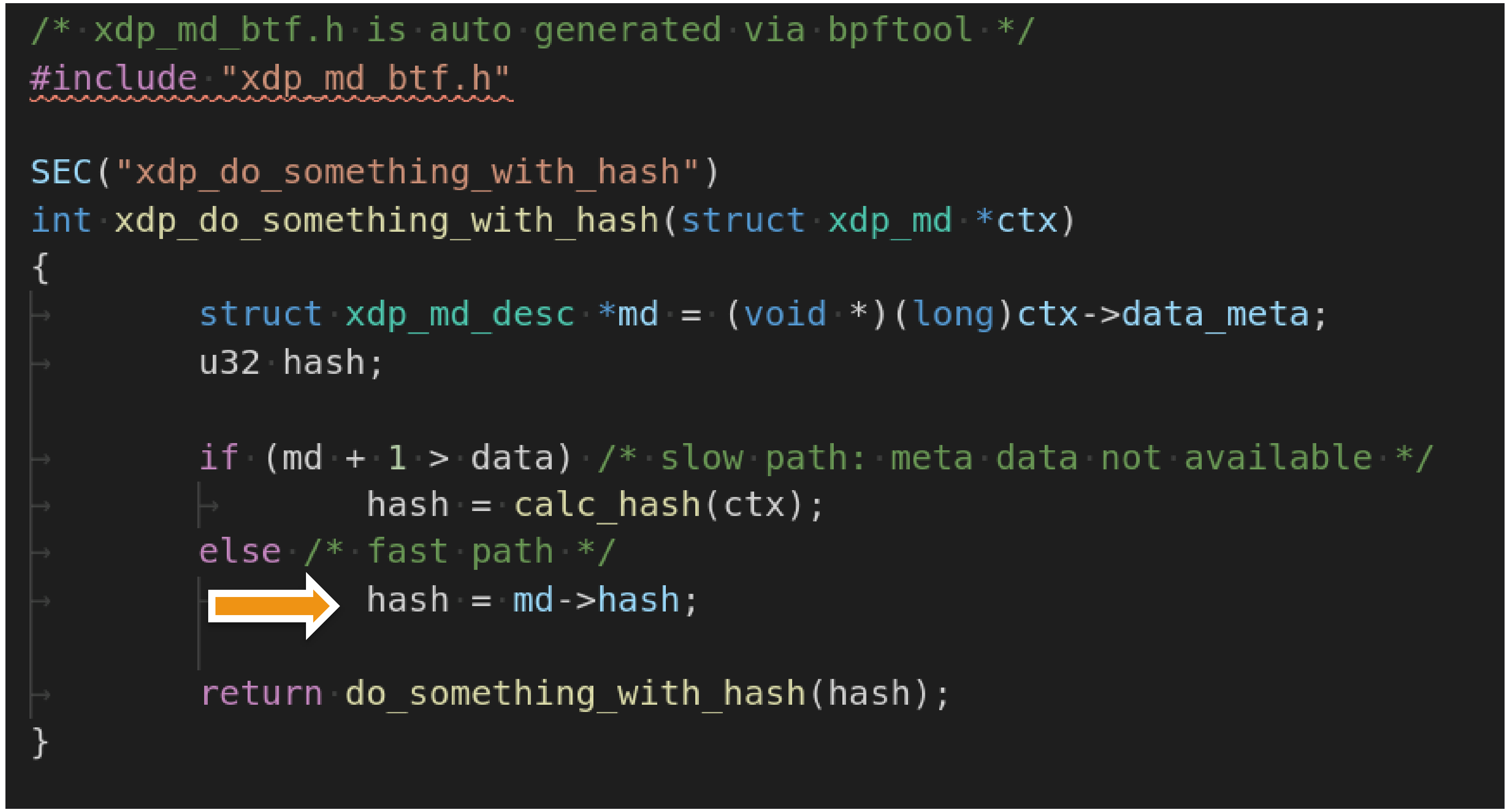
Example user program:

- Use the generated xdp_md_btf.h from driver:
  $ /usr/local/sbin/bpftool net xdp md_btf cstyle dev mlx0

- Access md using struct xdp_md_desc
  md->hash;
  md->vlan;
  md->flow_mark;
  whatever was advertise by driver:
  even md->mlx5_specific_offload :)

- Load the program and if XDP md is enabled the program will go to fast path

- In the example calc_hash() is very expensive (header parsing and computing the hash is cpu intensive and we can avoid it by simply taking the hw hash:
  - hash = md->hash;



```
/* xdp_md_btf.h is auto generated via bpftool */
#include "xdp_md_btf.h"

SEC("xdp_do_something_with_hash")
int xdp_do_something_with_hash(struct xdp_md *ctx)
{
        struct xdp_md_desc *md = (void *)(long)ctx->data_meta;
        u32 hash;

        if (md + 1 > data) /* slow path: meta data not available */
                hash = calc_hash(ctx);
        else /* fast path */
                hash = md->hash;

        return do_something_with_hash(hash);
}
```

Accelerated Kernel examples

- xdp_sample_pkts:
  - print packet meta data along with the sampled packet date
- xdp_redirect_cpu:
  - This example redirects packets to different CPUs (RSS Scaling using XDP)
  - Modified to use hw hash for redirecting instead of manual calculation of packet hash
- xdp_tx_iptunnel:
  - This example was written by FB to mimic katran, lookup specific ip and udp/tcp port, encapsulate in another ip header and send
  - Accelerated with TC flow mark to avoid ip and udp/tcp port lookup
- To compile these examples with meta data support, the driver specific header is required on run time
  - A new compilation flag was introduced to compile with metadata support:
  - XDP_MD_BTF=1 make M=samples/bpf
  - See next slide

Caution *

- Having the meta data fields sitting exactly before the actual packet buffer (xdp→data) is ok,
- **BUT:**

- Wen bpf_xdp_adjust_head is required (header expansion),
  - memmove(meta_data) will be required to preserve the meta data hints, causing a performance hit.

- Possible Solutions
  - 1) Invalidate meta data once consumed, this will break chaining:
    - bpf_xdp_adjust_meta(xdp, sizeof(*md));
  - 2) Place meta data starting at xdp_buff.data_hard_start, complicated..

XDP kernel example compile guide

- Install bpf samples from kernel source

```
$ make -C samples/bpf clean

$ make headers_install
```

- Compile samples with XDP metadata support

```
$ XDP_MD_BTF=1 make  M=samples/bpf

1 error generated.

  CLANG-bpf  samples/bpf/xdp_sample_pkts_kern.o

samples/bpf/xdp_sample_pkts_kern.c:60:10: fatal
error: 'xdp_md_btf.h' file not found
```

- fail due to missing driver md header file

```
# to generate it :

/usr/local/sbin/bpftool net xdp md_btf cstyle dev mlx0
> samples/bpf/xdp_md_btf.h
```

- Now again:

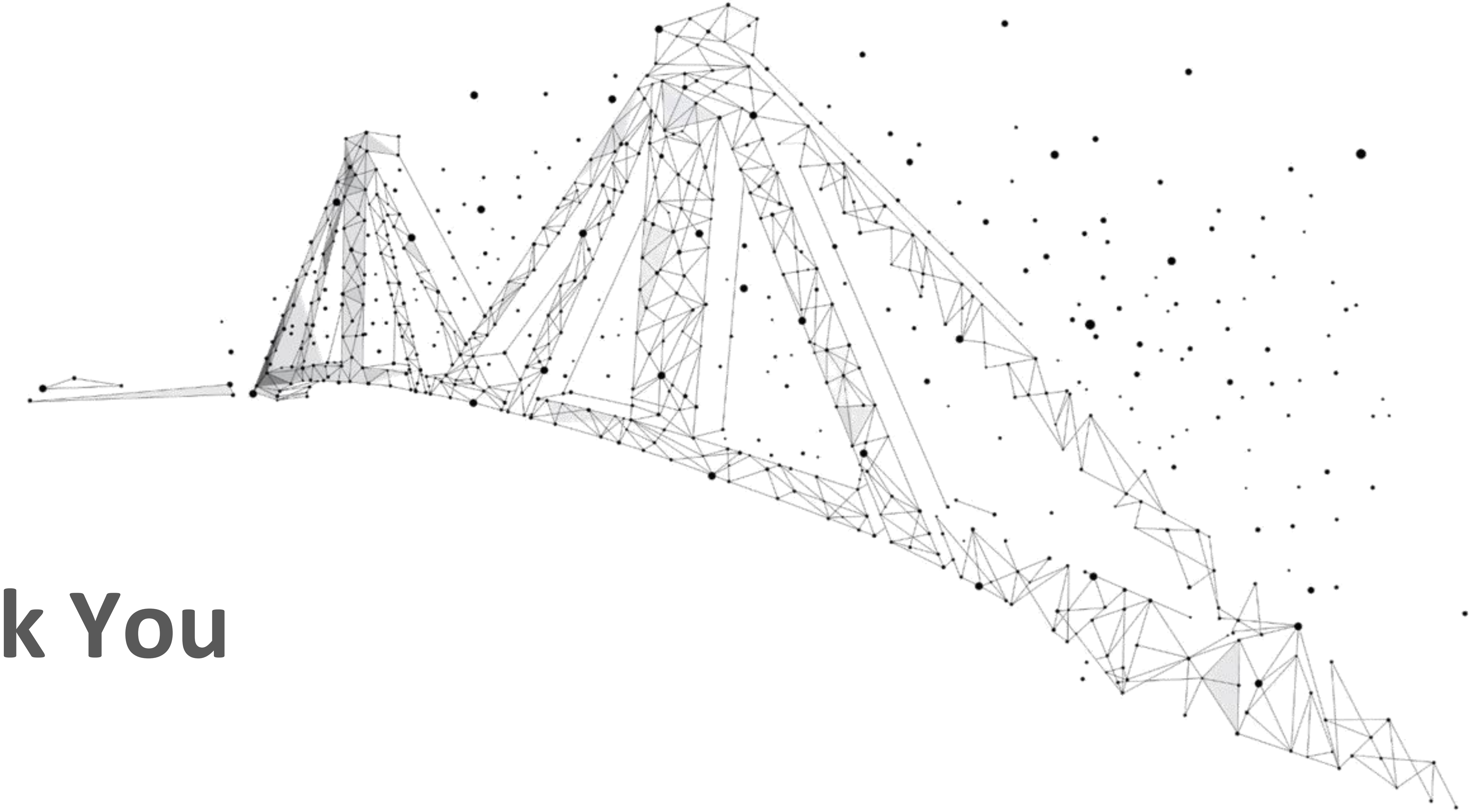
```
$ XDP_MD_BTF=1 make  M=samples/bpf
```

- Run Samples:

```
$ /usr/local/sbin/bpftool net xdp set dev
mlx0 md_btf on

$ ./samples/bpf/xdp_sample_pkts mlx0

Found xdp md prog

Flow mark: 0x2a, hash32: 0x46bb6ac7,
Ethernet hdr: 52 54 00 00 00 02 f6 a3 6e 59 16
83 08 00
```

Next ?

- Define TX hints usage
  - Just like RX but in the other direction
  - Separate md BTF for tx
  - XDP programs fills the “md_tx” struct, driver/HW consumes…
- Define the list of standard offloads
  - Design the kernel mechanism to enforce and validate them
- Program load time adaption to the current NIC’s BTF
  - To avoid recompiling the program per NIC/vendor
  - Reorder the md fields to the right offsets
  - Correct md fields references and accesses in the binary code
  - Restructure the binary code to avoid branching when an offloads/hint is not supported
- AF_XDP



Thank You