Integrating XFRM into XDP

Steffen Klassert

secunet Security Networks AG

Dresden

Linux Netconf, Boston, June, 2019

Why XFRM with XDP?

Highlevel design goals

Challenges

The proposed packet path

Open questions

XDP: eXpress Data Path

- XDP is a (in kernel) free programmable network stack
- Can be used to bypass parts of the kernel network stack
- Can program which parts of the network stack is needed
- Only the really needed parts of the network stack are called
- Should give more flexibility and better performance
- So make XFRM one of these programmable parts!

XDP: eXpress Data Path

- XDP is a (in kernel) free programmable network stack
- Can be used to bypass parts of the kernel network stack
- Can program which parts of the network stack is needed
- Only the really needed parts of the network stack are called
- Should give more flexibility and better performance
- So make XFRM one of these programmable parts!

XDP: eXpress Data Path

- XDP is a (in kernel) free programmable network stack
- Can be used to bypass parts of the kernel network stack
- Can program which parts of the network stack is needed
- Only the really needed parts of the network stack are called
- Should give more flexibility and better performance
- So make XFRM one of these programmable parts!

XDP: eXpress Data Path

- XDP is a (in kernel) free programmable network stack
- Can be used to bypass parts of the kernel network stack
- Can program which parts of the network stack is needed
- Only the really needed parts of the network stack are called
- Should give more flexibility and better performance
- So make XFRM one of these programmable parts!

XDP: eXpress Data Path

- XDP is a (in kernel) free programmable network stack
- Can be used to bypass parts of the kernel network stack
- Can program which parts of the network stack is needed
- Only the really needed parts of the network stack are called
- Should give more flexibility and better performance
- So make XFRM one of these programmable parts!

XDP: eXpress Data Path

- XDP is a (in kernel) free programmable network stack
- Can be used to bypass parts of the kernel network stack
- Can program which parts of the network stack is needed
- Only the really needed parts of the network stack are called
- Should give more flexibility and better performance
- So make XFRM one of these programmable parts!

XDP: eXpress Data Path

- XDP is a (in kernel) free programmable network stack
- Can be used to bypass parts of the kernel network stack
- Can program which parts of the network stack is needed
- Only the really needed parts of the network stack are called
- Should give more flexibility and better performance
- So make XFRM one of these programmable parts!

Some design goals

- Should work as a forwarding fastpath
- Should skip everything in the stack that is not needed
- Fastpath flows should be configured without user interaction
- Should use a 'eBPF flow hashmap' to cache flows
- Don't reimplement XFRM in eBPF XDP $\rightarrow$ use existing XFRM stack

Some design goals

- Should work as a forwarding fastpath
- Should skip everything in the stack that is not needed
- Fastpath flows should be configured without user interaction
- Should use a 'eBPF flow hashmap' to cache flows
- Don't reimplement XFRM in eBPF XDP $\rightarrow$ use existing XFRM stack

Some design goals

- Should work as a forwarding fastpath
- Should skip everything in the stack that is not needed
- Fastpath flows should be configured without user interaction
- Should use a 'eBPF flow hashmap' to cache flows
- Don't reimplement XFRM in eBPF XDP $\rightarrow$ use existing XFRM stack

Some design goals

- Should work as a forwarding fastpath
- Should skip everything in the stack that is not needed
- Fastpath flows should be configured without user interaction
- Should use a 'eBPF flow hashmap' to cache flows
- Don't reimplement XFRM in eBPF XDP → use existing XFRM stack

Some design goals

- Should work as a forwarding fastpath
- Should skip everything in the stack that is not needed
- Fastpath flows should be configured without user interaction
- Should use a 'eBPF flow hashmap' to cache flows
- Don't reimplement XFRM in eBPF XDP $\rightarrow$ use existing XFRM stack

Some design goals

- Should work as a forwarding fastpath
- Should skip everything in the stack that is not needed
- Fastpath flows should be configured without user interaction
- Should use a 'eBPF flow hashmap' to cache flows
- Don't reimplement XFRM in eBPF XDP → use existing XFRM stack

Challenges on integrating XFRM into XDP

- Hooks in very early at L2 (before allocating a sk_buff)
- No sk_buff!!!
- packet representation in XDP with struct xdp_buff

Challenges on integrating XFRM into XDP

- Hooks in very early at L2 (before allocating a sk_buff)
- No sk_buff!!!
- packet representation in XDP with struct xdp_buff

Challenges on integrating XFRM into XDP

- Hooks in very early at L2 (before allocating a sk_buff)
- No sk_buff!!!
- packet representation in XDP with struct xdp_buff

Challenges on integrating XFRM into XDP

- Hooks in very early at L2 (before allocating a sk_buff)
- No sk_buff!!!
- packet representation in XDP with struct xdp_buff

Packet representation in XDP:

```
1 struct xdp_buff {
2       void *data;
3       void *data_end;
4       void *data_meta;
5       void *data_hard_start;
6       unsigned long handle;
7       struct xdp_rxq_info *rxq;
8 };
```

- data: Pointer to the start of the packet data
- data_end: Pointer to the end of the packet data
- data_meta: Pointer to optional metadata (max. 32 byte)
- data_hard_start: Pointer to maximum possible headroom
- handle: new???
- rxq: Pointer to an internal receive queue metadata structure

Packet representation in XDP:

```
1 struct xdp_buff {
2       void *data;
3       void *data_end;
4       void *data_meta;
5       void *data_hard_start;
6       unsigned long handle;
7       struct xdp_rxq_info *rxq;
8 };
```

- data: Pointer to the start of the packet data
- data_end: Pointer to the end of the packet data
- data_meta: Pointer to optional metadata (max. 32 byte)
- data_hard_start: Pointer to maximum possible headroom
- handle: new???
- rxq: Pointer to an internal receive queue metadata structure

Packet representation in XDP:

```
1 struct xdp_buff {
2       void *data;
3       void *data_end;
4       void *data_meta;
5       void *data_hard_start;
6       unsigned long handle;
7       struct xdp_rxq_info *rxq;
8 };
```

- data: Pointer to the start of the packet data
- data_end: Pointer to the end of the packet data
- data_meta: Pointer to optional metadata (max. 32 byte)
- data_hard_start: Pointer to maximum possible headroom
- handle: new???
- rxq: Pointer to an internal receive queue metadata structure

Packet representation in XDP:

```
1 struct xdp_buff {
2       void *data;
3       void *data_end;
4       void *data_meta;
5       void *data_hard_start;
6       unsigned long handle;
7       struct xdp_rxq_info *rxq;
8 };
```

- data: Pointer to the start of the packet data
- data_end: Pointer to the end of the packet data
- data_meta: Pointer to optional metadata (max. 32 byte)
- data_hard_start: Pointer to maximum possible headroom
- handle: new???
- rxq: Pointer to an internal receive queue metadata structure

Packet representation in XDP:

```
1 struct xdp_buff {
2         void *data;
3         void *data_end;
4         void *data_meta;
5         void *data_hard_start;
6         unsigned long handle;
7         struct xdp_rxq_info *rxq;
8 };
```

- data: Pointer to the start of the packet data
- data_end: Pointer to the end of the packet data
- data_meta: Pointer to optional metadata (max. 32 byte)
- data_hard_start: Pointer to maximum possible headroom
- handle: new???
- rxq: Pointer to an internal receive queue metadata structure

Packet representation in XDP:

```
1 struct xdp_buff {
2       void *data;
3       void *data_end;
4       void *data_meta;
5       void *data_hard_start;
6       unsigned long handle;
7       struct xdp_rxq_info *rxq;
8 };
```

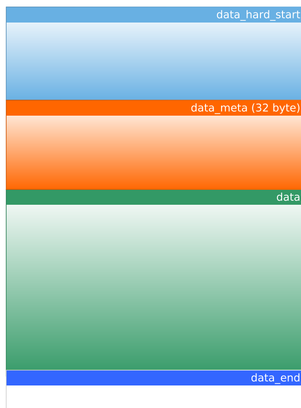
- data: Pointer to the start of the packet data
- data_end: Pointer to the end of the packet data
- data_meta: Pointer to optional metadata (max. 32 byte)
- data_hard_start: Pointer to maximum possible headroom
- handle: new???
- rxq: Pointer to an internal receive queue metadata structure

Packet representation in XDP:

```
1 struct xdp_buff {
2       void *data;
3       void *data_end;
4       void *data_meta;
5       void *data_hard_start;
6       unsigned long handle;
7       struct xdp_rxq_info *rxq;
8 };
```

- data: Pointer to the start of the packet data
- data_end: Pointer to the end of the packet data
- data_meta: Pointer to optional metadata (max. 32 byte)
- data_hard_start: Pointer to maximum possible headroom
- handle: new???
- rxq: Pointer to an internal receive queue metadata structure

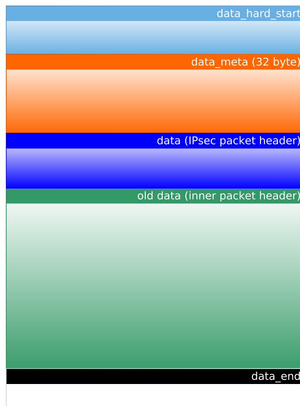
xdp_buff packet layout



Packet data is always linear, following inequation holds

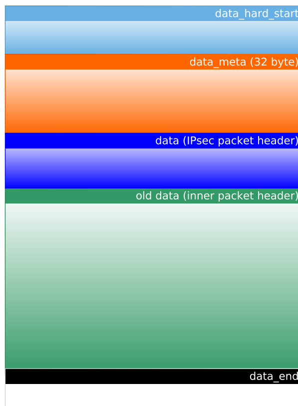
$$data_hard_start <= data_meta <= data < data_end$$

xdp_buff IPsec packet layout



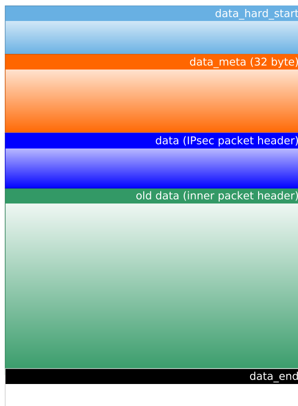
- Need space for the IPsec trailer
- We have 32 byte metadata but need 64 byte metadata

xdp_buff IPsec packet layout



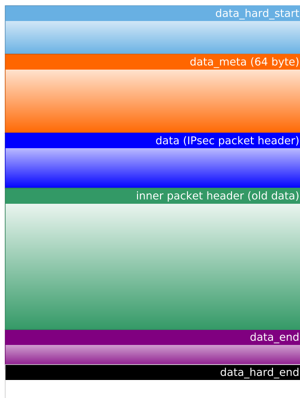
- Need space for the IPsec trailer
- We have 32 byte metadata but need 64 byte metadata

xdp_buff IPsec packet layout



- Need space for the IPsec trailer
- We have 32 byte metadata but need 64 byte metadata

xdp_buff IPsec packet layout



$$data_hard_start <= data_meta <= data < data_end <= data_hard_end$$

Adjusted packet representation in XDP:

```
1 struct xdp_buff {
2       void *data;
3       void *data_end;
4       void *data_meta;
5       void *data_hard_start;
6       void *data_hard_end;
7       unsigned long handle;
8       struct xdp_rxq_info *rxq;
9 };
```

- data: Pointer to the start of the packet data
- data_end: Pointer to the end of the packet data
- data_meta: Pointer to optional metadata (max. 64 byte)
- data_hard_start: Pointer to maximum possible headroom
- data_hard_end: Pointer to maximum possible tailroom
- handle: new???
- rxq: Pointer to an internal receive queue metadata structure

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

Challenges on integrating XFRM into XDP

- BPF helpers have to be created to call XFRM functions
- Problem: Most XFRM code relies on having a sk_buff
- Two options:
  - Convert XFRM to not use sk_buff
    - Use common metdata structure
    - Advantage: No code duplication
    - Disadvantage: Easy to introduce bugs
  - Create new xfrm_xdp callbacks
    - Advantage: Standard XFRM is not touched
    - Disadvantage: Creates lot of new code
  - Need to choose one option before we start implementing

The proposed packet path

- Distinguish known and unknown flows
- A flow is known if it has an entry in a 'eBFP flow hashmap'
- Unknown flows (first packet) go to standard network stack
- Known flows go to the XDP fastpath

The proposed packet path

- Distinguish known and unknown flows
- A flow is known if it has an entry in a 'eBFP flow hashmap'
- Unknown flows (first packet) go to standard network stack
- Known flows go to the XDP fastpath

The proposed packet path

- Distinguish known and unknown flows
- A flow is known if it has an entry in a 'eBFP flow hashmap'
- Unknown flows (first packet) go to standard network stack
- Known flows go to the XDP fastpath

The proposed packet path

- Distinguish known and unknown flows
- A flow is known if it has an entry in a 'eBFP flow hashmap'
- Unknown flows (first packet) go to standard network stack
- Known flows go to the XDP fastpath

The proposed packet path

- Distinguish known and unknown flows
- A flow is known if it has an entry in a 'eBFP flow hashmap'
- Unknown flows (first packet) go to standard network stack
- Known flows go to the XDP fastpath

First packet of a flow

- No match in the XDP eBFP flow hashmap
- It takes a full round through the stack
- Can be seen as the 'configuration packet'
- Inserts flow informations into the eBFP flow hasmap
  - Only flows that are really forwarded are inserted
  - Local input packets don't insert flows
  - Dropped packets don't insert flows
  - No violation of the systems security policy

First packet of a flow

- No match in the XDP eBFP flow hashmap
- It takes a full round through the stack
- Can be seen as the 'configuration packet'
- Inserts flow informations into the eBFP flow hasmap
  - Only flows that are really forwarded are inserted
  - Local input packets don't insert flows
  - Dropped packets don't insert flows
  - No violation of the systems security policy

First packet of a flow

- No match in the XDP eBFP flow hashmap
- It takes a full round through the stack
- Can be seen as the 'configuration packet'
- Inserts flow informations into the eBFP flow hasmap
  - Only flows that are really forwarded are inserted
  - Local input packets don't insert flows
  - Dropped packets don't insert flows
  - No violation of the systems security policy

First packet of a flow

- No match in the XDP eBFP flow hashmap
- It takes a full round through the stack
- Can be seen as the 'configuration packet'
- Inserts flow informations into the eBFP flow hasmap
    - Only flows that are really forwarded are inserted
    - Local input packets don't insert flows
    - Dropped packets don't insert flows
    - No violation of the systems security policy

First packet of a flow

- No match in the XDP eBFP flow hashmap
- It takes a full round through the stack
- Can be seen as the 'configuration packet'
- Inserts flow informations into the eBFP flow hasmap
  - Only flows that are really forwarded are inserted
  - Local input packets don't insert flows
  - Dropped packets don't insert flows
  - No violation of the systems security policy

First packet of a flow

- No match in the XDP eBFP flow hashmap
- It takes a full round through the stack
- Can be seen as the 'configuration packet'
- Inserts flow informations into the eBFP flow hasmap
  - Only flows that are really forwarded are inserted
  - Local input packets don't insert flows
  - Dropped packets don't insert flows
  - No violation of the systems security policy

First packet of a flow

- No match in the XDP eBFP flow hashmap
- It takes a full round through the stack
- Can be seen as the 'configuration packet'
- Inserts flow informations into the eBFP flow hasmap
  - Only flows that are really forwarded are inserted
  - Local input packets don't insert flows
  - Dropped packets don't insert flows
  - No violation of the systems security policy

First packet of a flow

- No match in the XDP eBFP flow hashmap
- It takes a full round through the stack
- Can be seen as the 'configuration packet'
- Inserts flow informations into the eBFP flow hasmap
  - Only flows that are really forwarded are inserted
  - Local input packets don't insert flows
  - Dropped packets don't insert flows
  - No violation of the systems security policy

First packet of a flow

- No match in the XDP eBFP flow hashmap
- It takes a full round through the stack
- Can be seen as the 'configuration packet'
- Inserts flow informations into the eBFP flow hasmap
  - Only flows that are really forwarded are inserted
  - Local input packets don't insert flows
  - Dropped packets don't insert flows
  - No violation of the systems security policy

Subsequent packets of a flow

- Match in the XDP eBFP flow hasmap
- Flow valid: Apply IPsec and send it out directly
- Flow invalid: Full round through the stack

Subsequent packets of a flow

- Match in the XDP eBFP flow hasmap
- Flow valid: Apply IPsec and send it out directly
- Flow invalid: Full round through the stack

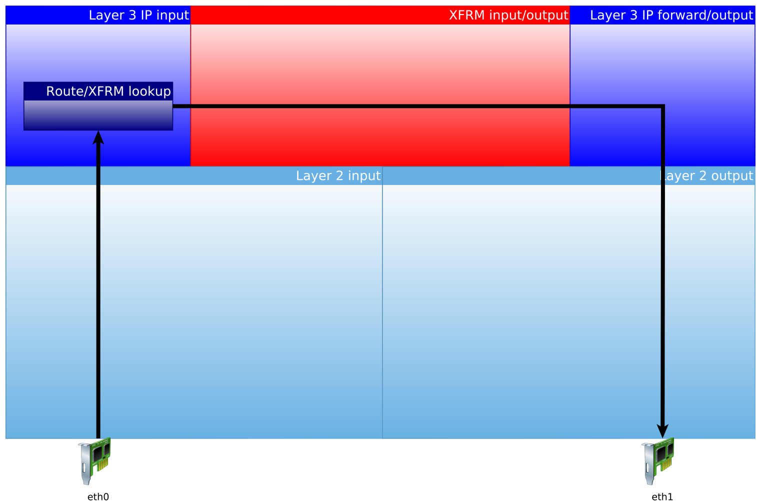
Subsequent packets of a flow

- Match in the XDP eBFP flow hasmap
- Flow valid: Apply IPsec and send it out directly
- Flow invalid: Full round through the stack

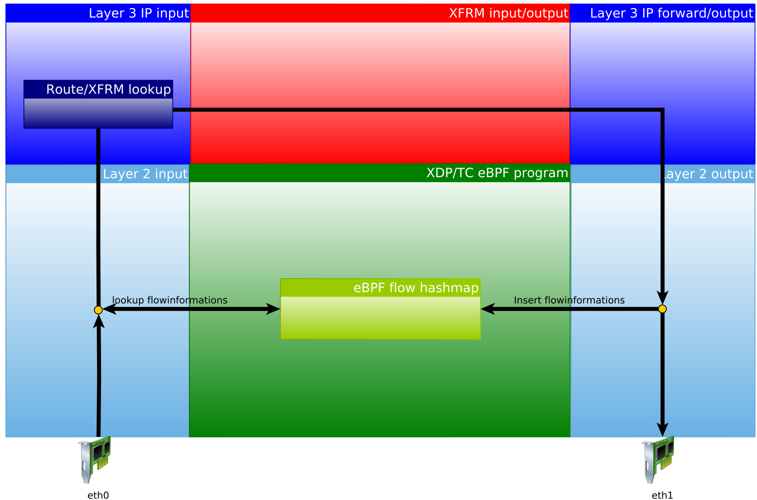
Subsequent packets of a flow

- Match in the XDP eBFP flow hasmap
- Flow valid: Apply IPsec and send it out directly
- Flow invalid: Full round through the stack

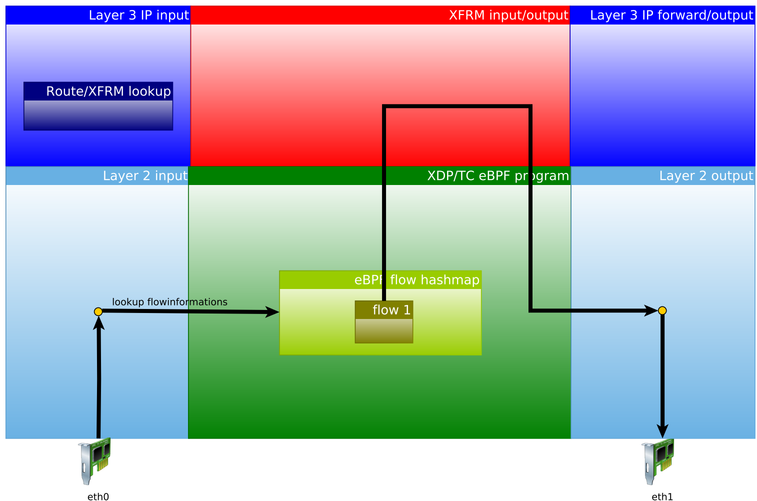
Standard forwarding with XFRM



XDP: First packet



XDP: Subsequent packets



Open questions

- How to handle asynchronous returns from the crypto layer?
- XDP is usefull for forwarding, what about local input?
  - Maybe we need some early eBPF TC hook for the sk_buff case
    - Could be used to cache flow informations
    - Maybe parts of the local input stack can be bypassed too

Open questions

- How to handle asynchronous returns from the crypto layer?
- XDP is usefull for forwarding, what about local input?
  - Maybe we need some early eBPF TC hook for the sk_buff case
    - Could be used to cache flow informations
    - Maybe parts of the local input stack can be bypassed too

Open questions

- How to handle asynchronous returns from the crypto layer?
- XDP is usefull for forwarding, what about local input?
  - Maybe we need some early eBPF TC hook for the sk_buff case
    - Could be used to cache flow informations
    - Maybe parts of the local input stack can be bypassed too

Open questions

- How to handle asynchronous returns from the crypto layer?
- XDP is usefull for forwarding, what about local input?
  - Maybe we need some early eBPF TC hook for the sk_buff case
    - Could be used to cache flow informations
    - Maybe parts of the local input stack can be bypassed too

Open questions

- How to handle asynchronous returns from the crypto layer?
- XDP is usefull for forwarding, what about local input?
  - Maybe we need some early eBPF TC hook for the sk_buff case
    - Could be used to cache flow informations
    - Maybe parts of the local input stack can be bypassed too

Open questions

- How to handle asynchronous returns from the crypto layer?
- XDP is usefull for forwarding, what about local input?
  - Maybe we need some early eBPF TC hook for the sk_buff case
    - Could be used to cache flow informations
    - Maybe parts of the local input stack can be bypassed too