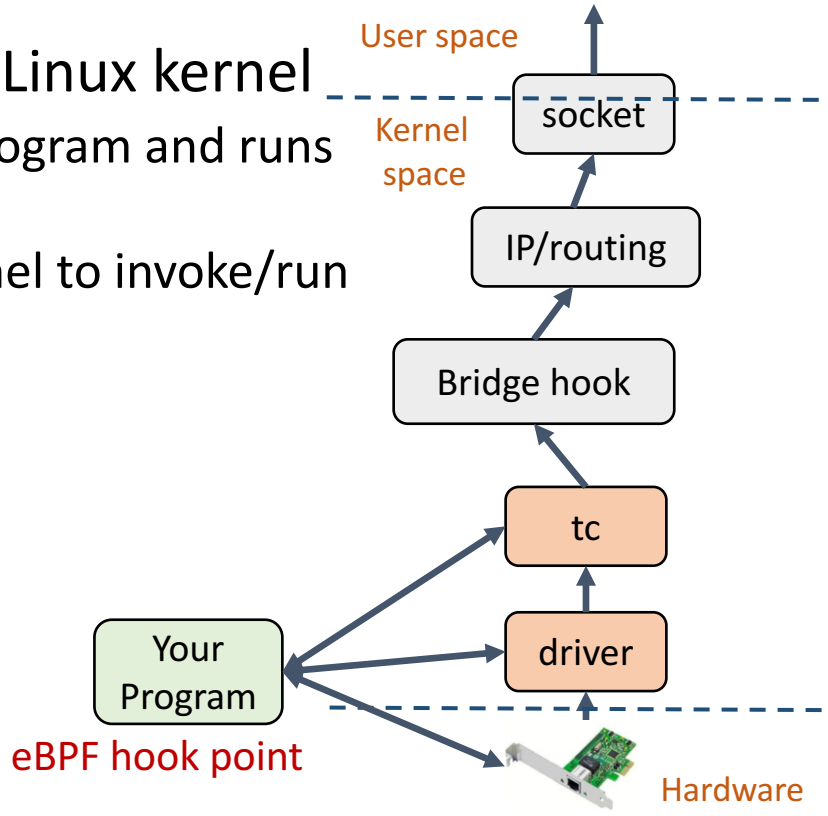
P4C-XDP: Programming the Linux Kernel Forwarding Plane using P4

William Tu, VMware NSBU
Mihai Budiu, VMware Research
{mbudiu,tuc}@vmware.com

June 5, 2018
P4 Workshop

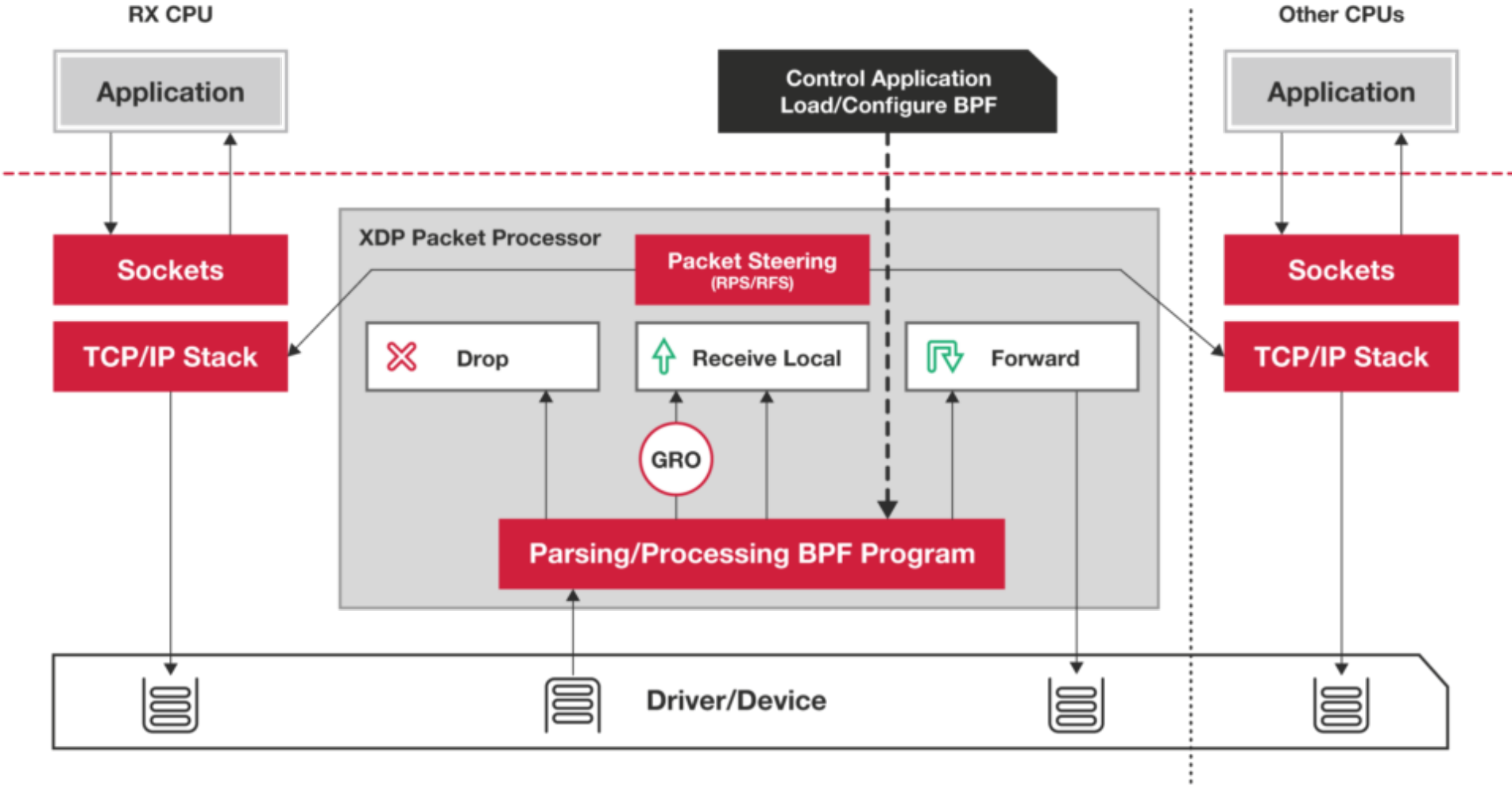
What is eBPF / XDP ?

- A virtual machine running in Linux kernel
  - A way to write a restricted C program and runs in Linux kernel
  - A set of hook points inside kernel to invoke/run the BPF program
- Benefits
  - Extensible
  - Safe
  - Fast



Example of TC+eBPF

XDP: eXpress Data Path



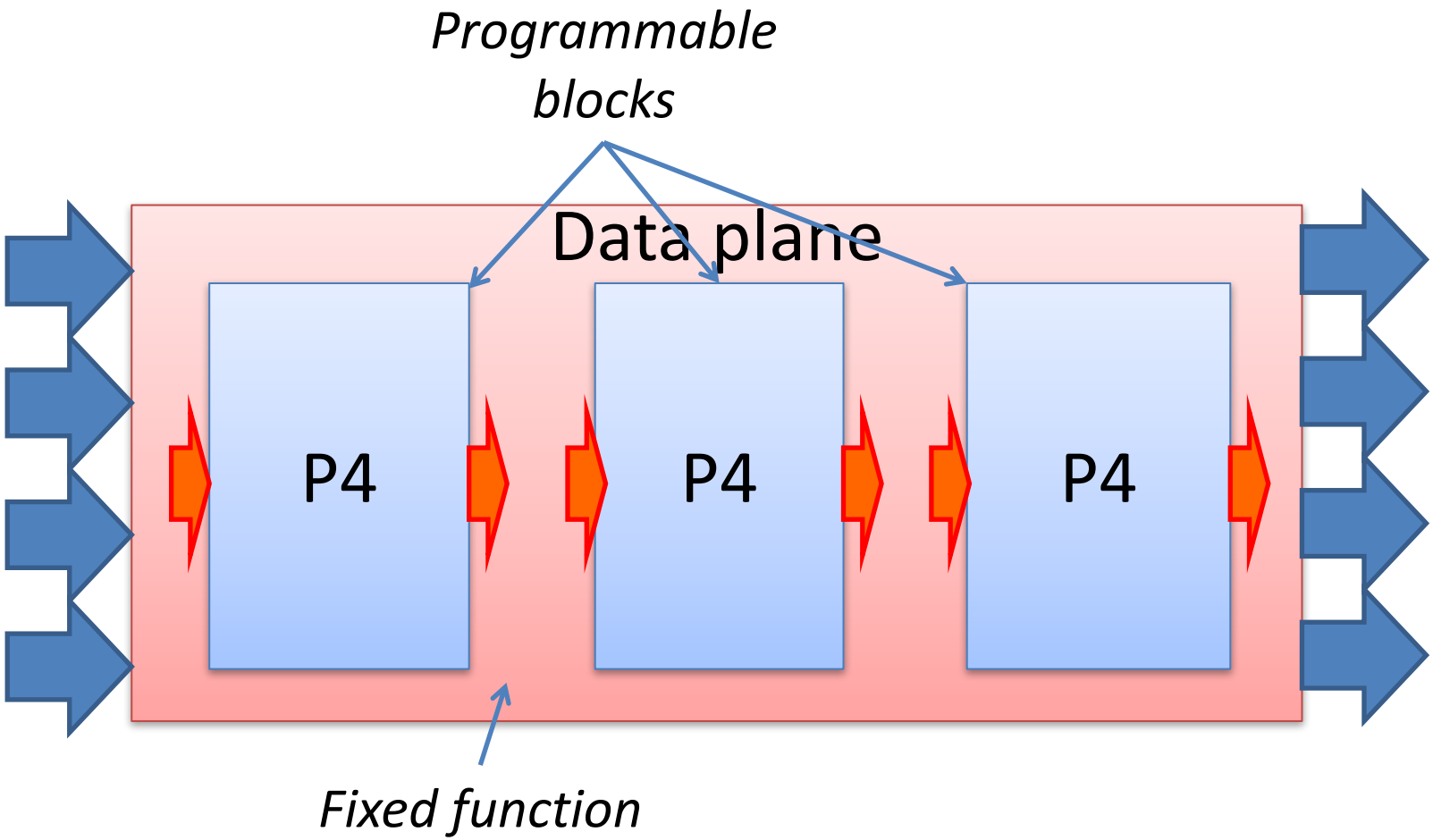
https://www.iovisor.org/technology/xdp

$P4_{16}$-> C -> eBPF

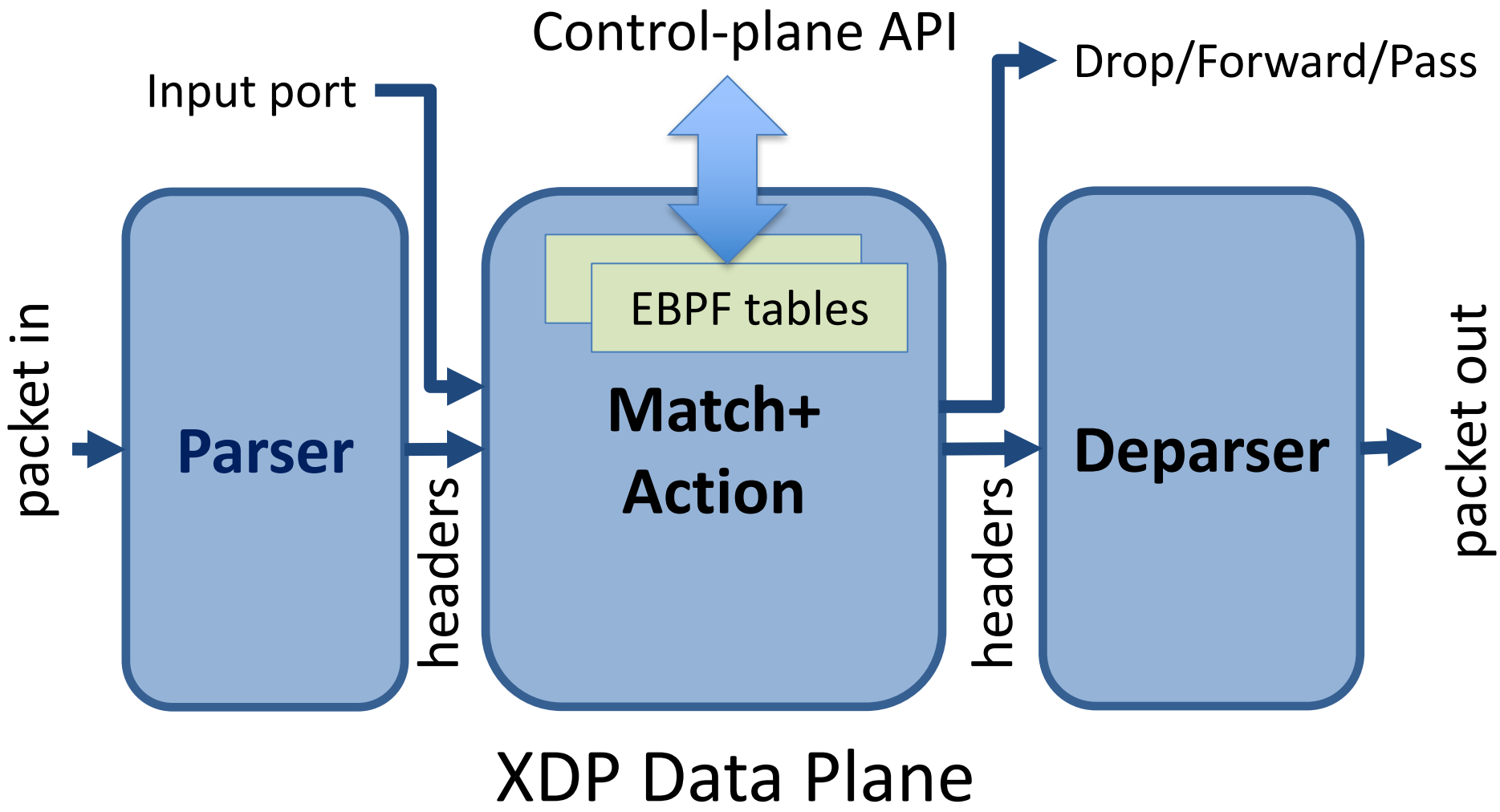
- p4c-xdp: back-end for the $P4_{16}$ reference compiler

- Generate stylized C
  - Filtering, forwarding, encapsulation
  - No loops, all data on stack
  - eBPF tables for control/data-plane communication
  - LLVM can generate eBPF bytecode

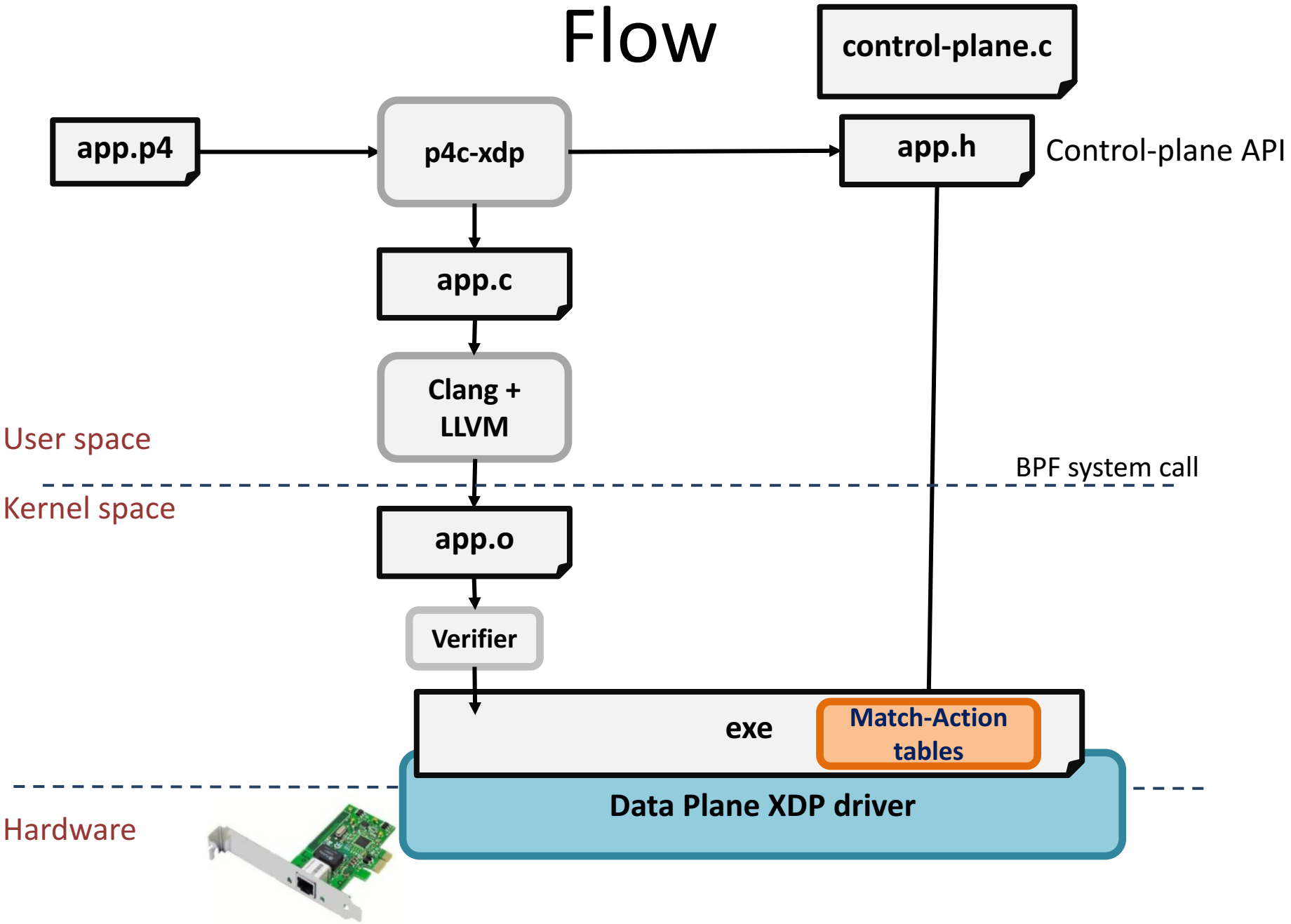
$P4_{16}$ Generic Data Plane Model



The XDP Switching Model

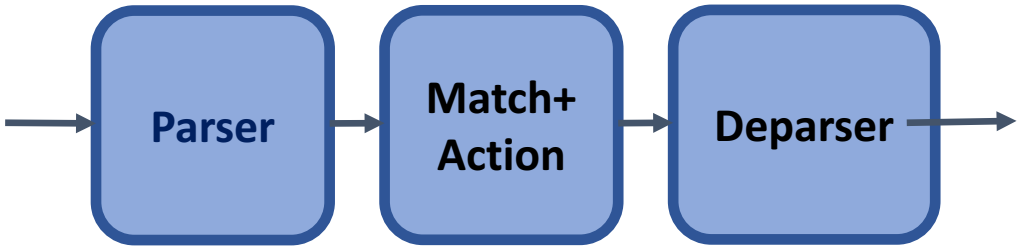


Flow


control-plane.c
app.p4
p4c-xdp
app.h
Control-plane API
app.c
Clang +
LLVM
User space
BPF system call
Kernel space
app.o
Verifier
exe
Match-Action
tables
Data Plane XDP driver
Hardware

Simple Example

- Parse Ethernet and IPv4 header
- Lookup a table using Ethernet's destination as key
- Based on Ethernet's destination address, execute two actions:
  - Drop the packet (XDP_DROP)
  - Pass the packet to network stack (XDP_PASS)



Protocol Header Definition

```
header Ethernet {
        bit<48> source;
        bit<48> destination;
        bit<16> protocol;
}
header IPv4{
        bit<4> version;
        bit<4> ihl;
        bit<8> diffserv;
        …
}
```

```
struct Headers {
        Ethernet ethernet;
        IPv4    ipv4;
}
```

xdp.h

```
struct Ethernet{
          u8 source[6];
          u8 destination[6];
          u16 protocol;
          u8 ebpf_valid;
}
```

C struct + valid bit

P4 Protocol Parser

```
parser Parser(packet_in packet,
              out Headers hd) {
    state start {
        packet.extract(hd.ethernet);
        transition select(hd.ethernet.protocl) {
            16w0x800: parse_ipv4;
            default: accept;
        }
    }
    state parse_ipv4 {
        packet.extract(hd.ipv4);
        transition accept;
    }
}
```

Code Block

Direct Pkt Access

Switch-case

goto

Code Block

Direct Pkt Access

P4: Table Match and Action

```
control Ingress (inout Headers hdr,
                 in xdp_input xin, out xdp_output xout) {
      action Drop_action() {
            xout.output_action = xdp_action.XDP_DROP; }
      action Fallback_action() {
            xout.output_action = xdp_action.XDP_PASS; }

      table mactable {
            key = {hdr.ethernet.destination : exact; }
            actions = {
                  Fallback_action;
                  Drop_action;
            }
            implementation = hash_table(64);
       }
```

Two action types

BPF HashMap

Key size of 6 byte

Value with enum type + parameter

XDP C code: xdp1.c

```
SEC("prog")
int ebpf_filter(struct xdp_md *skb) {
      struct Headers hd = {};
      …
      /* parser */
      if (end < start + header_size)
            goto reject;
      hd.ethernet.destination[0] = load_byte(…);
      …
      /* match+action*/
      value = bpf_map_lookup_elem(key);
      switch(value->action) {
            case Drop_action:
            …
      /* deparser */
      xdp_adjust_head();
      // update packet header
      return xout.xdp_output;
```

Generate Header for Control Plane

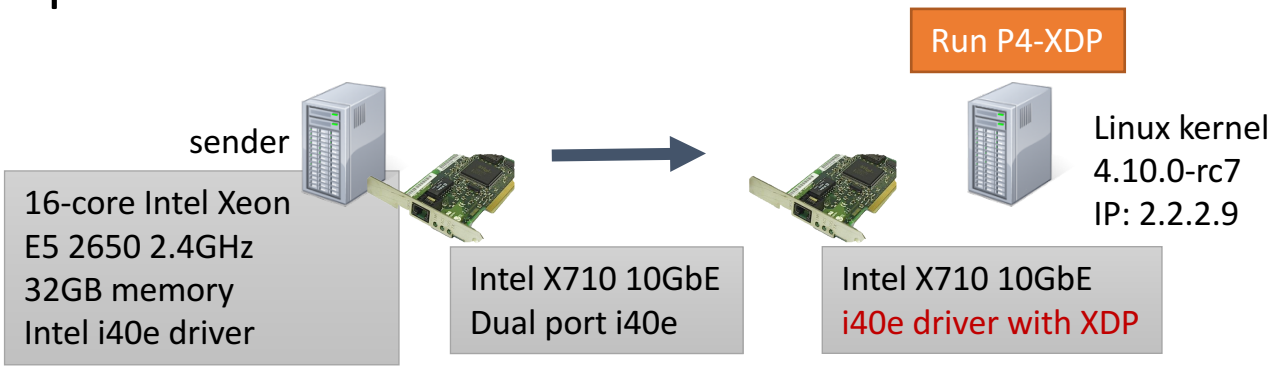
Generate: xpd1.h

```
struct mactable_key {
            u8 field0[6];
}
enum mactable_actions {
            Fallback_action,
            Drop_action,
}
struct mactable_value {
            enum mactable_actions action;
            union {
                        struct {
                        } Fallback_action;
                        struct {
                        } Drop_action;
            } u;
}
```

User provide: user_xpd1.c

```
#include ”xdp1.h”
int main () {
        int fd = bpf_obj_get(MAP_PATH);
        …
        struct mactable_key key;
        memcpy(key.field0, MACADDR, 6);
        struct mactable_value value;
        value.action = Fallback_action;
        // Add a new entry to the table
        bpf_update_elem(fd, &key, &value, BPF_ANY);
}
```

Setup and Installation



- Source code at Github (Apache License)
  - https://github.com/vmware/p4c-xdp
  - Vagrant box / docker image available
- Dependencies:
  - P4 2016: https://github.com/p4lang/p4c
  - Linux >= 4.12: http://www.kernel.org/
  - iproute2 >= 4.8.0: https://www.kernel.org/pub/linux/utils/net/iproute2/
  - clang+LLVM >=3.7.1: http://llvm.org/releases
- P4C-XDP binary
  - #./p4c-xdp --target xdp -o <**output_file**> <**input p4**>

Demo1: Swap Ethernet (xdp11.p4)

- Swap Ethernet source and destination
- Send to the receiving interface (return **XDP_TX**)

```
bit<48> tmp;
apply {
    if (hd.ipv4.isValid())
    {
        tmp = hd.ethernet.destination;
        hd.ethernet.destination = hd.ethernet.source;
        hd.ethernet.source = tmp;
    }
```

https://youtu.be/On7hEJ6bPVU

https://github.com/vmware/p4c-xdp/blob/master/tests/xdp11.p4

Thank You

Questions?

https://github.com/vmware/p4c-xdp

Demo2: ping4/6 and stats (xdp12.p4)

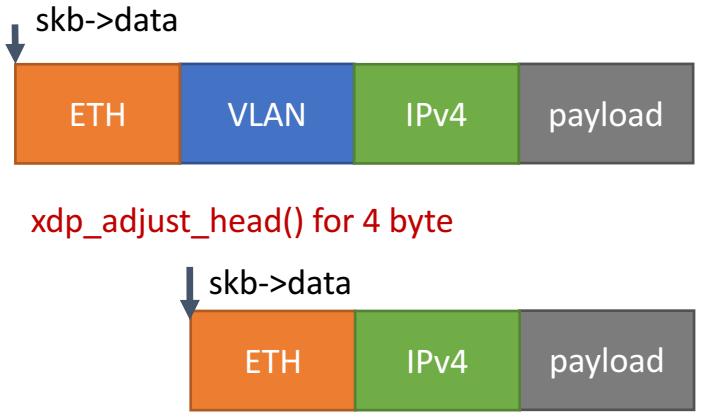
- Parse IPv4/IPv6 ping
- Drop ipv6 ping, and return **XDP_DROP**
- Enable Control plane
- Update ipv4 statistics, and return **XDP_PASS**

https://youtu.be/vlp1MzWVOc8

Deparser: Update the Packet

```
control Deparser(in Headers hdrs,
                 packet_out packet) {
        apply {
                packet.emit(hdrs.ethernet);
                packet.emit(hdrs.vlan_tag);
                packet.emit(hdrs.ipv4);
        }
}
```

Example: VLAN pop



The payload remains in the same memory

- Users can push/pop headers by emitting more or skipping emit
  - Ex: vlan push/pop by add/remove packet.emit(hdrs.vlan_tag);
  - Need to adjust skb->data by adding xdp_adjust_head helper

xdp_model.p4

```
enum xdp_action {
    XDP_ABORTED,  // some fatal error occurred during processing;
    XDP_DROP,     // packet should be dropped
    XDP_PASS,     // packet should be passed to the Linux kernel
    XDP_TX        // packet resent out on the same interface
}

struct xdp_input {
    bit<32> input_port;
}

struct xdp_output {
    xdp_action output_action;
    bit<32> output_port;  // output port for packet
}

parser xdp_parse<H>(packet_in packet, out H headers);
control xdp_switch<H>(inout H hdrs, in xdp_input i, out xdp_output o);
control xdp_deparse<H>(in H headers, packet_out packet);
```

P4-XDP: xdp1.c

```
SEC("prog")
int ebpf_filter(struct xdp_md *skb) {
        struct Headers hd = {};
        …
        /* parser */
        if (end < start + header_size)
                goto reject;
        hd.ethernet.destination[0] = load_byte(…);
        …
        /* match+action*/
        value = bpf_map_lookup_elem(key);
        switch(value->action) {
                case Drop_action:
                …
        /* deparser */
        xdp_adjust_head();
        // update packet header
        return xout.xdp_output;
```

- Parser:
  - Check packet access boundary.
  - Walk through the protocol graph.
  - Save in "struct Headers hd."
- Match+Action:
  - Extract key from struct Headers
  - Lookup BPF hash map
  - Execute the correponding action
- Deparser
  - Convert headers back into a byte stream.
  - Only valid headers are emitted.