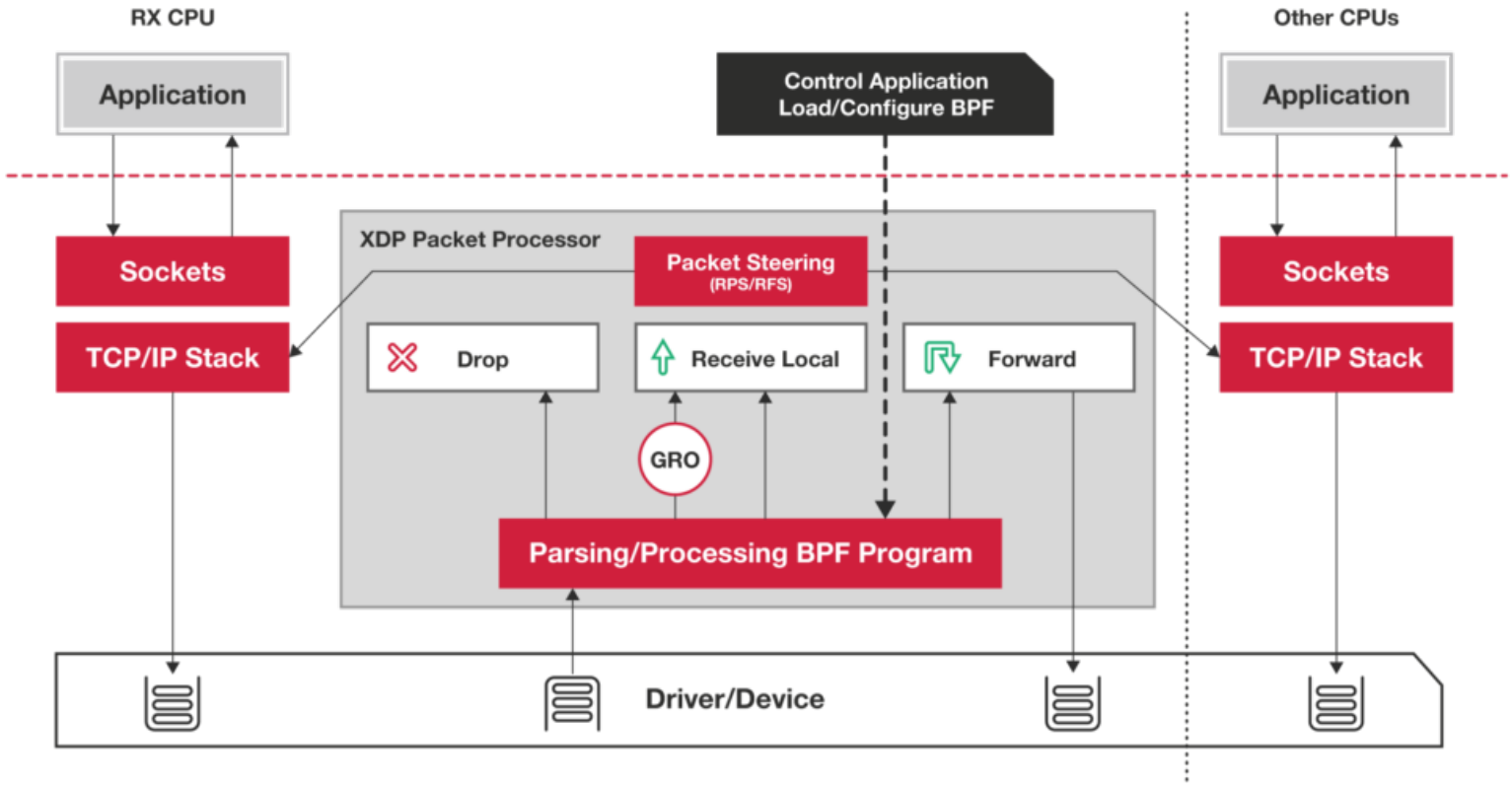
Compiling P4 to XDP

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

February 27, 2017
IOVisor summit

XDP



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

P4

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, David Walker *ACM SIGCOMM Computer Communications Review (CCR). Volume 44, Issue #3 (July 2014)*

Initially designed for programmable switches. $P4_{16}$ can support many kinds of packet processing devices.

http://P4.org

P4.org Consortium

Carriers, cloud operators, chip, networking, systems, universities, start-ups

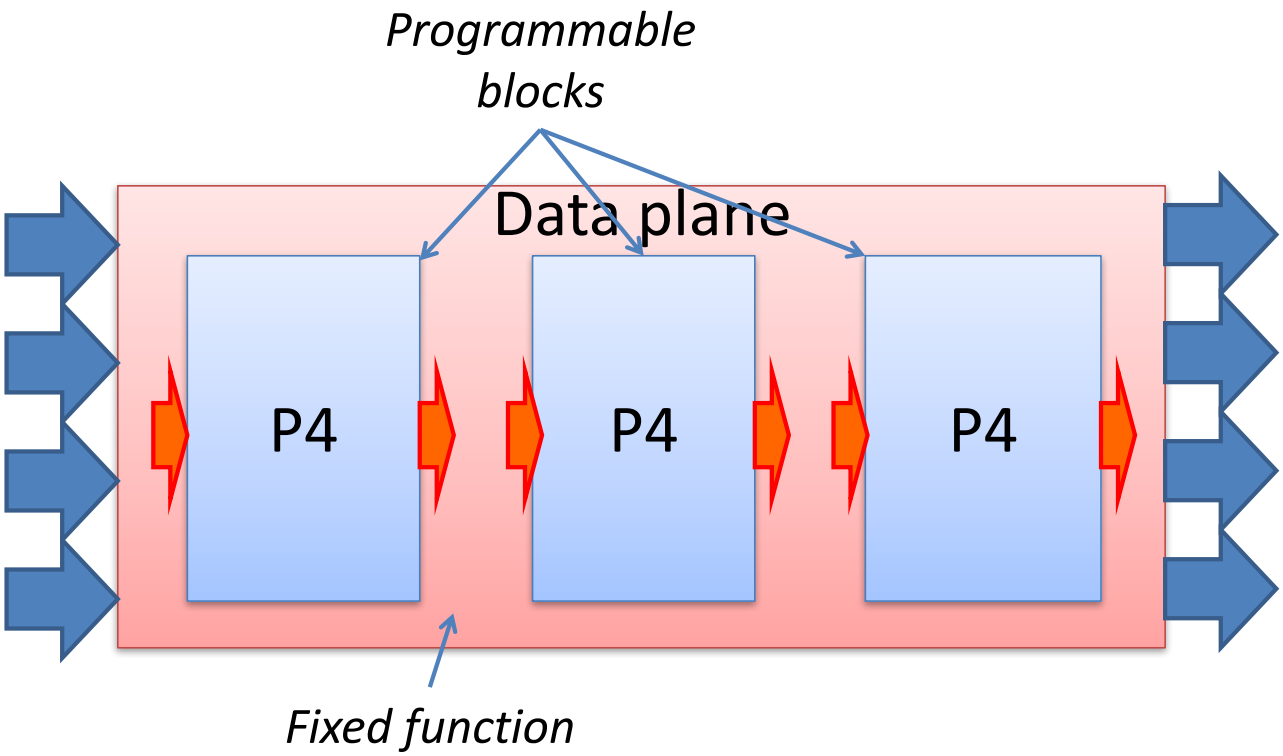
$P4_{16}$

- C-like, strongly typed
- Arbitrary length bitstrings
- Match-action tables
- Parser = state machine
- No loops, no pointers, no memory allocation
- Support for external, target-specific accelerators (e.g., checksum units, multicast, learning, etc.)

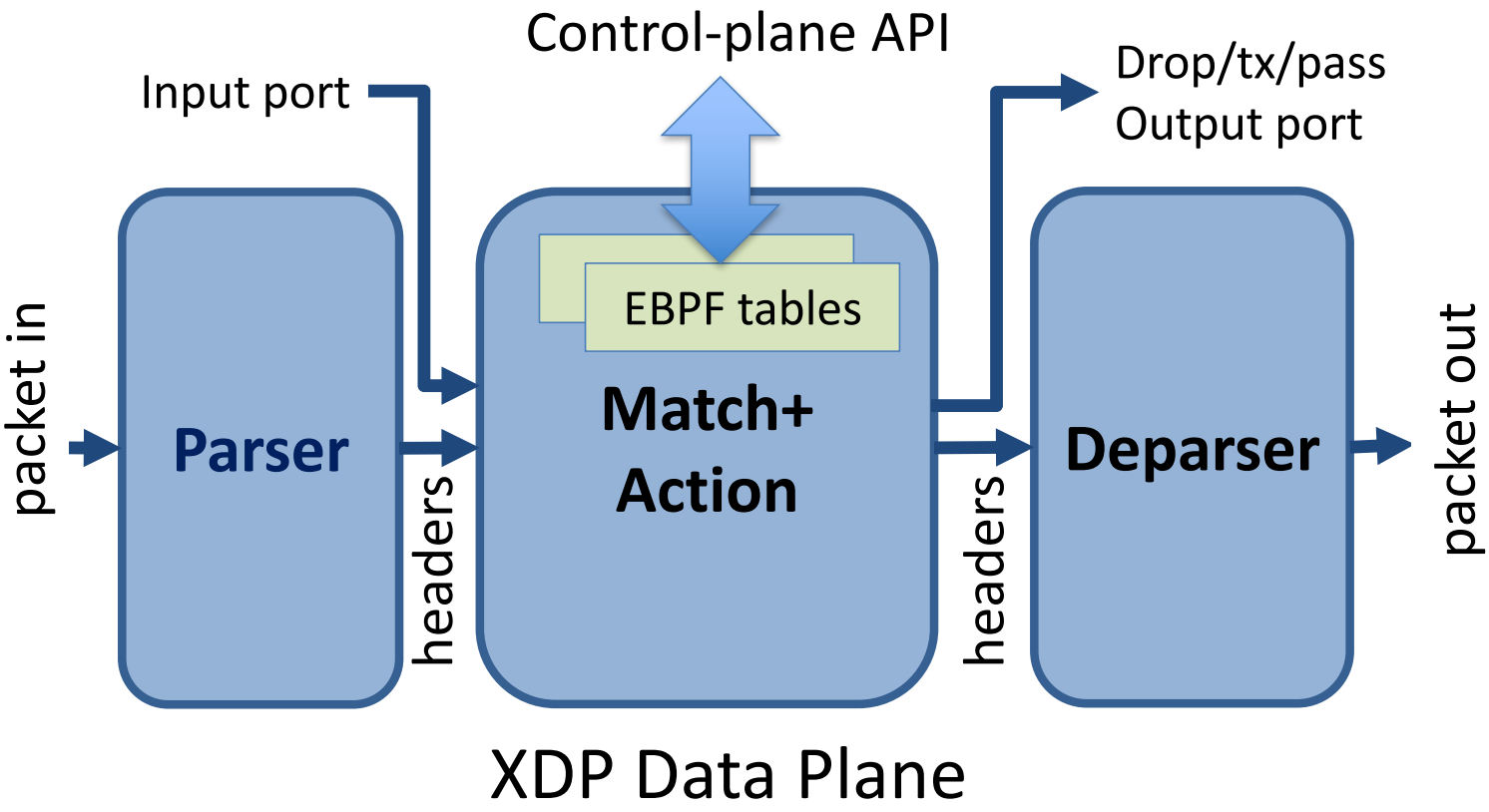
$P4_{16}$-> C -> EBPF

- p4c-xdp: back-end for the $P4_{16}$ reference compiler
- Generate stylized C
  - No loops, all data on stack
  - EBPF tables for control/data-plane communication
  - Filtering, forwarding, encapsulation
  - Currently use Linux TC subsystem for forwarding
- https://github.com/williamtu/p4c-xdp

$P4_{16}$ generic data plane model

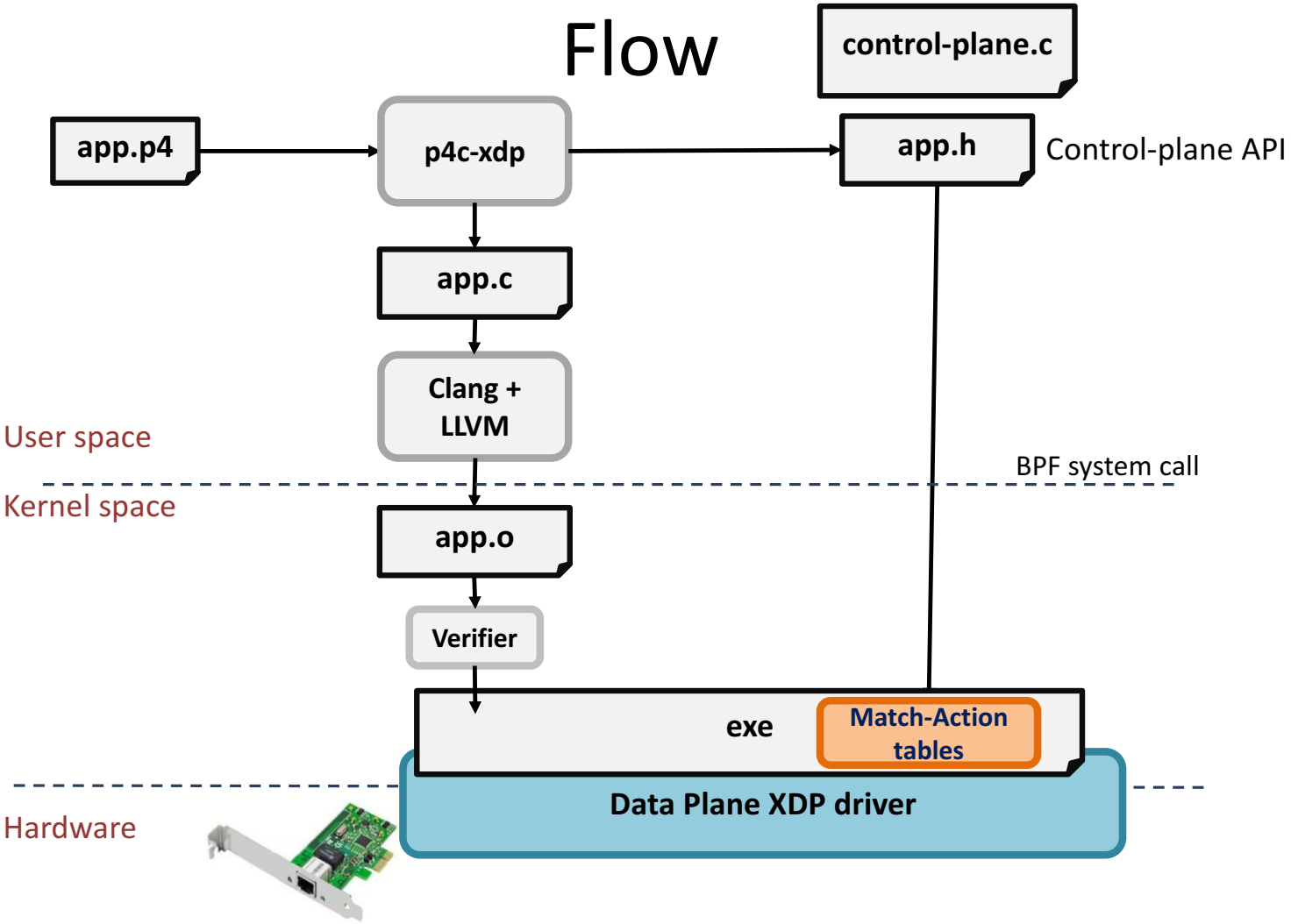


The XDP switching model



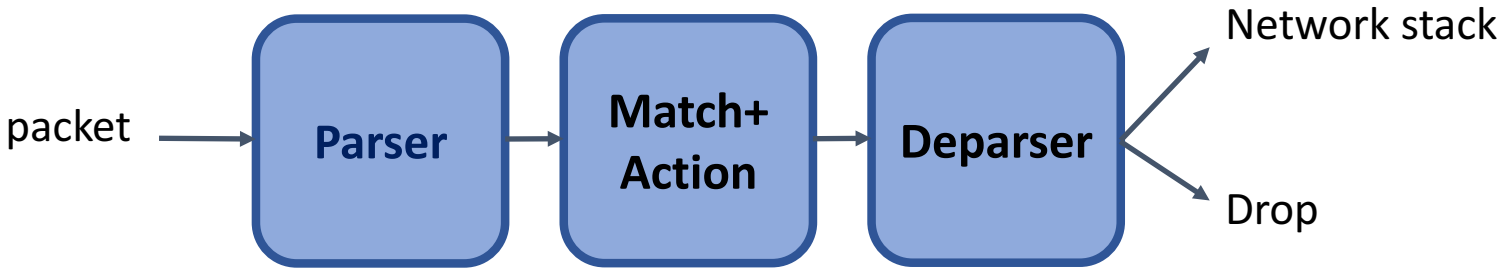
xdp_model.p4

```
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);

package xdp<H>(xdp_parse<H> p, xdp_switch<H> s, xdp_deparse<H> d);
```


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 one **action**:
  - Drop the packet (**XDP_DROP**)
  - Pass the packet to network stack (**XDP_PASS**)



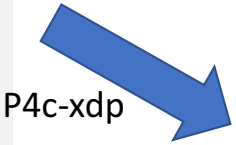
P4 Protocol Header Definition

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

C struct + valid bit

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

C struct



xdp.h

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

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

Code Block

BPF Direct Pkt Access

Switch-case

goto

BPF Direct Pkt Access

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

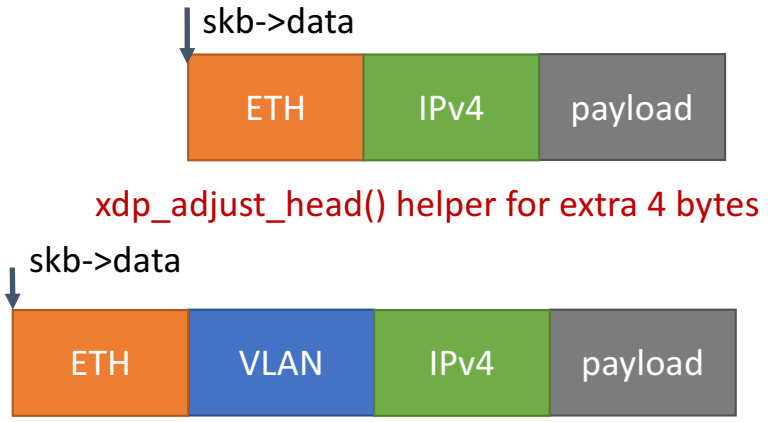
Key size of 6 byte

Value with enum type + parameter

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 Push



The payload remains in the same memory

- Parser saves results at '**hdrs**'
- 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

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 strea
  - Only valid headers are emitted.

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;

        bpf_update_elem(fd, &key, &value,
BPF_ANY);

}
```

Setup and Installation

- Source code at Github
  - git clone https://github.com/williamtu/p4c-xdp/
  - Vagrant box / docker image available
- Dependencies:
  - P4 2016: https://github.com/p4lang/p4c
  - Linux >= 4.10.0-rc7: 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_xdp.c**> <**input.p4**>

Experiences with BPF Verifier

- Typical packet access check: **data** + [**off**] <= **data_end**
  - where [off] can be either immediate or
  - coming from a tracked register that contains an immediate

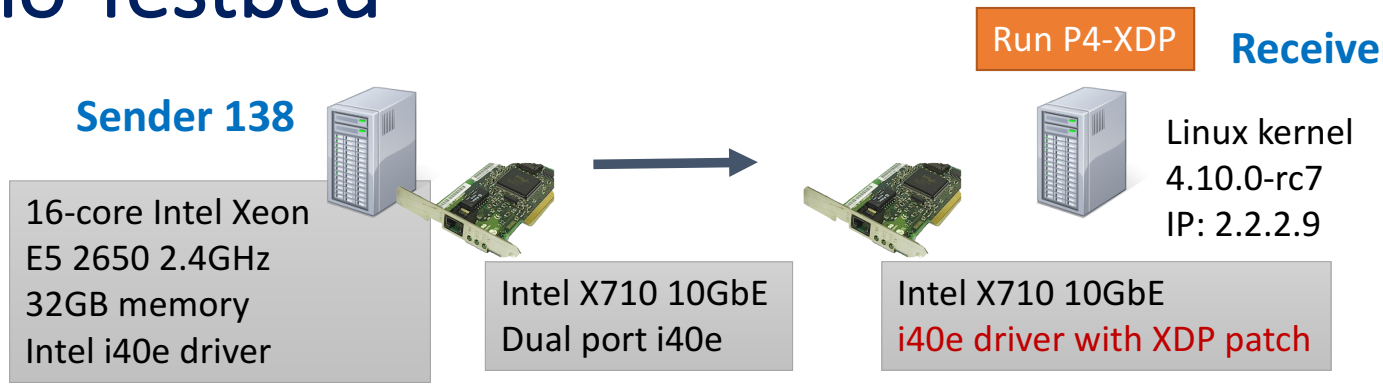
```
R1=pkt(id=0,off=0,r=22) R2=pkt_end  R3=imm144,min_value=144,max_value=144
30: (bf) r5 = r3
31: (07) r5 += 23
32: (77) r5 >>= 3
33: (bf) r6 = r1      // r6 == pkt
34: (0f) r6 += r5     // pkt += r5
```

- Two patches related to direct packet access
  - bpf: enable verifier to better track const **alu ops**, commit 3fadc8011583
  - bpf: enable verifier to add 0 to packet ptr, commit 63dfef75ed753

Pending Issues

- **BPF 512 Byte maximum stack size** [#22]
  - Not necessarily due to the size of local variables
  - LLVM allocates too many things into 8 byte registers
  - LLVM spills registers onto the stack
  - Possible workarounds:
    - Bump up the maximum stack size in kernel
    - Enable more efficient use of stack in LLVM
- **Registers having const_imm spills without tracking state** [#34]
  - BPF only has 10 registers, LLVM spills the register to stack when necessary
  - BPF verifier keeps the register states and restore after BPF_LOAD
  - Current version does not support spill const_imm

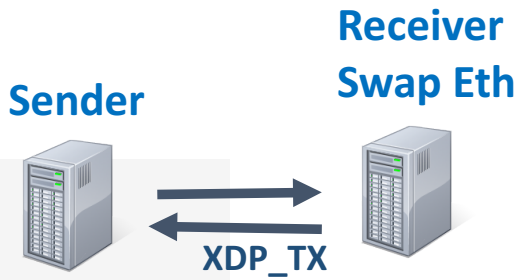
Demo Testbed



- Linux kernel net-next 4.10.0-rc7
  - Due to two BPF verifier fixes
  - Plus our own 2 patches to increase BPF stack size to 4096
- i40e XDP driver
  - V4 patch: http://patchwork.ozlabs.org/patch/706701/
- Demo source code at, see demo*
  - https://github.com/williamtu/p4c-xdp/tree/master/tests/

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

https://youtu.be/On7hEJ6bPVU

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

- Parse IPv4/IPv6 ICMP ping
- Drop ipv6 ping, and return **XDP_DROP**
- Demonstrate control plane
- Update ipv4 statistics, and return **XDP_PASS**

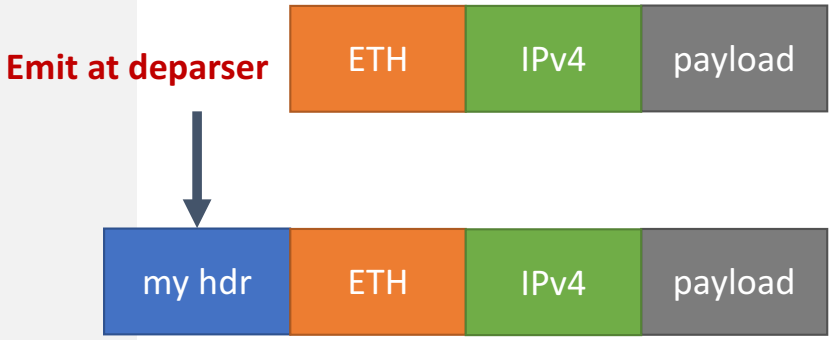
https://github.com/williamtu/p4c-xdp/blob/master/tests/xdp12.p4

https://youtu.be/vlp1MzWVOc8

Demo3: Encapsulation (xdp16.p4)

- Define a customized header
- Insert the header in front of Ethernet (or any where you want)

```
header myhdr_t
{
   bit<32> id;
   bit<32> timestamp;
}
control Ingress(…) {
   action TS_action()
   {
      hd.myhdr.ts = BPF_KTIME_GET_NS();  // BPF helper
      hd.myhdr.id = 0xfefefefe;
      xoutdrop = false; //XDP_PASS
   }
```



https://github.com/williamtu/p4c-xdp/blob/master/tests/xdp16.p4

https://youtu.be/TibGxCXPNVc

Future Wok

- Forward / broadcast /clone
  - Currently rely on TC (bpf_skb_clone_redirect)
  - XDP_FORWARD support in driver/kernel?
- Recirculation
  - Add recirculation support in XDP driver
  - Return xdp_recirculate and tail call.
- Use cases

Thank You

Questions?