Linux Network Programming with P4

Linux Plumbers 2018

Fabian Ruffy, William Tu, Mihai Budiu

VMware Inc. and University of British Columbia
Outline

• Introduction to P4
• XDP and the P4 Compiler
• Testing
• Example
• Performance Results
• Discussion

Fabian
William
What is P4?

• High-level programming language for network data planes
  • Allows for protocol flexibility
  • Specifies a packet processing pipeline
• Compiled and loaded into target platform
• Open and standardized

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)
P4 Essentials

• C-like, strongly typed language
• Type and memory-safe (no pointers)
• Bounded execution (no loops)
• Statically allocated (no malloc, no recursion)
• Spec: 
  http://github.com/p4lang/p4-spec
• Reference compiler implementation:
  http://github.com/p4lang/p4c (Apache 2 license)
P4 Software Workflow

P4 Architecture Model

P4 Compiler

Supplied

Target
P4 Software Workflow

- P4 Program
- P4 Architecture Model
- P4 Compiler

User-supplied

Supplied

Target
P4 Software Workflow

- P4 Program
- P4 Compiler
- P4 Architecture Model
- Target

User-supplied

Supplied
P4 Software Workflow

- **User-supplied**
  - P4 Program
  - P4 Architecture Model

- **Supplied**
  - Target-specific configuration binary

- **Compiled**
  - P4 Compiler

- **Target**
P4 Software Workflow

- P4 Program
- P4 Architecture Model
- P4 Compiler
- Target-specific configuration binary

Load

- Tables
- Extern objects

Data Plane

Target

User-supplied

Supplied
P4 Software Workflow

- **P4 Architecture Model**
- **P4 Program**
- **P4 Compiler**
- **Target**
- **User-supplied**
- **Control Plane**
- **Data Plane**
- **Supplied**
- **Load**
- Tables
- Extern objects
P4 Software Workflow

P4 Program → P4 Compiler → Control Plane

User-supplied

P4 Architecture Model

Target-specific configuration binary

Supplied

RUNTIME

Control Plane

Add/remove table entries
Extern control
Packet-in/out

Data Plane

Tables
Extern objects
CPU port

Target

Load

RUNTIME
P4\textsubscript{16} generic data-plane model

- \texttt{os\_lib.p4}
- \texttt{switch\_lib.p4}
- \texttt{npu\_lib.p4}
- \texttt{nic\_lib.p4}

The networking stack of the OS

\texttt{program.p4}
P4 and XDP
eBPF/XDP

• Virtual machine running in the Linux kernel
• Provides:
  • The ability to write restricted C and run it in the kernel
  • A set of kernel hook points invoking the eBPF program
• Extensible, safe and fast
• Alternative to user-space networking
eBPF/XDP

- Virtual machine running in the Linux kernel
- Provides:
  - The ability to write restricted C and run it in the kernel
  - A set of kernel hook points invoking the eBPF program
- Extensible, safe and fast
- Alternative to user-space networking

A programmable data plane in the Linux kernel!
# P4 vs eBPF/XDP

<table>
<thead>
<tr>
<th>Feature</th>
<th>P4</th>
<th>eBPF/XDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Safe</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Safety</td>
<td>Type system</td>
<td>Verifier</td>
</tr>
<tr>
<td>Loops</td>
<td>In parsers</td>
<td>Tail calls (dynamic limit)</td>
</tr>
<tr>
<td>Resources</td>
<td>Statically allocated</td>
<td>Statically allocated</td>
</tr>
<tr>
<td>Policies</td>
<td>Tables (match+action)</td>
<td>Maps (tables)</td>
</tr>
<tr>
<td>Extern helpers</td>
<td>Target-specific</td>
<td>Hook-specific</td>
</tr>
<tr>
<td>Control-plane API</td>
<td>Synthesized by compiler</td>
<td>eBPF maps</td>
</tr>
</tbody>
</table>
The P4 eBPF backends

- p4c-ebpf is part of the open-source distribution
  - [http://github.com/p4lang/p4c/backends/ebpf](http://github.com/p4lang/p4c/backends/ebpf)
- p4c-xdp is a separate open-source project
  - [http://github.com/vmware/p4c-xdp](http://github.com/vmware/p4c-xdp)
  - Extension of the p4c compiler
  - Reuses much of the code

- Not production-ready
  - Needs more work
  - Known bugs and limitations
  - Generated not efficient yet
Generating XDP code
$P4_{16} \rightarrow \text{C} \rightarrow \text{eBPF/XDP}$

- Generates stylized C
- No tail calls yet, all data on stack
- eBPF tables control/data-plane communication
- Can do filtering, forwarding, encapsulation
- Relies on Linux TC for forwarding
  - We plan on switching to libbpf
The XDP Switching Model

Parser

Match+
Action

Deparser

Input port

Control-plane API

Drop/Forward/Pass

EBPF tables

XDP Data Plane
Flow

User space

Kernel space

BPF system call

Hardware

exe

Match-Action tables

Data Plane XDP driver
Flow

- `app.p4` ➔ `p4c-xdp` ➔ `app.c`

**User space**
**Kernel space**

**Hardware**

BPF system call
Flow

- **app.p4**
- **p4c-xdp**
- **app.c**
- **Clang + LLVM**
- **app.o**
- **Verifier**
- **Data Plane XDP driver**

1. User space
2. Kernel space
3. Hardware

---

**BPF system call**

**Match-Action tables**
**Flow**

- **app.p4** ➔ **p4c-xdp** ➔ **app.c** ➔ **Clang + LLVM** ➔ **app.o** ➔ **Verifier** ➔ **exe**
- **control-plane.c** ➔ **app.h** ➔ **app.o**

- **User space**
- **Kernel space**
- **Hardware**

**Data Plane XDP driver**

**Match-Action tables**

**Control-plane API**

**BPF system call**
Testing P4-XDP code
Test Frameworks

• User-space testing
  • Isolates specification from implementation
  • Validates correctness of generated code
  • User-space wrappers around eBPF tables and APIs
  • Reads and writes packets from capture files

• Kernel-space testing
  • Loads eBPF program into kernel
  • I/O connected to virtual interfaces
  • Writes capture files to interfaces in user-space
  • Records output using tcpdump
Five Testing Stages
Five Testing Stages

1. compile p4
2. parse stf

test.p4

test.stf
Five Testing Stages

1. Compile `p4`
2. Parse `stf`

Runtime source

Input files:
- `input0.pcap`
- `input1.pcap`
- `...`

Expected outputs:
- `expect0.pcap`
- `expect1.pcap`
- `...`
Five Testing Stages

1. Compile p4
2. Parse stf
3. Compile data-plane

Input:
- input0.pcap
- input1.pcap
- expect0.pcap
- expect1.pcap

Output:
- runtime source

Files:
- test.p4
- test.stf
Five Testing Stages

1. compile p4
2. parse stf
3. compile data-plane
4. runtime source
5. runtime executable

- test.p4
- test.stf
- input0.pcap
- input1.pcap
- expect0.pcap
- expect1.pcap
Five Testing Stages
Five Testing Stages

1. **compile p4**
2. **parse stf**
3. **compile data-plane**
4. **run**
5. **check results**

- **test.p4**
- **input0.pcap**, **input1.pcap**, **…**
- **expect0.pcap**, **expect1.pcap**, **…**
- **output0.pcap**
- **output1.pcap**
Five Testing Stages

1. Compile
   - test.p4
   - compile
   - p4

2. Parse
   - test.stf
   - parse
   - stf

3. Compile Data-plane
   - runtime source
   - compile
data-plane

4. Run
   - runtime executable
   - run
   - output0.pcap
   - output1.pcap
   - …

5. Check Results
   - check
   - results
   - pass/fail

- Input:
  - input0.pcap
  - input1.pcap
  - …

- Expectations:
  - expect0.pcap
  - expect1.pcap
  - …
A sample P4-XDP program
Forwarding an IPv4 Packet

• 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 Headers

```c
header Ethernet {
    bit<48> source;
    bit<48> dest;
    bit<16> protocol;
}

header IPv4{
    bit<4> version;
    bit<4> ihl;
    bit<4> diffserv;
    ...
}

struct Headers {
    Ethernet eth;
    IPv4    ipv4;
}
```
P4 Headers

```c
header Ethernet {  
    bit<48> source;  
    bit<48> dest;  
    bit<16> protocol;  
}
header IPv4{  
    bit<4> version;  
    bit<4> ihl;  
    bit<8> diffserv;  
    ...  
}
struct Headers {  
    Ethernet eth;  
    IPv4 ipv4;  
}
```

```
struct Ethernet{  
    u8 source[6];  
    u8 destination[6];  
    u16 protocol;  
    u8 ebpf_valid;  
}
struct IPv4 {  
    u8 version[6]; /* bit<4> */  
    u8 ihl[6]; /* bit<4> */  
    u8 diffserv; /* bit<8> */  
}
```

- Currently each header field is re-aligned
- Inefficient design
P4 Protocol Parser

```c
parser Parser(packet_in packet, out Headers hd) {
    state start {
        packet.extract(hd.ethernet);
        transition select(hd.ethernet.protocol) {
            16w0x800: parse_ipv4;
            default: accept;
        }
    }
    state parse_ipv4 {
        packet.extract(hd.ipv4);
        transition accept;
    }
}
```
P4 Protocol Parser

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

```c
struct Headers hd = {};
...
if (end < start + header_size)
    goto reject;
hd.ethernet.destination[0] = load_byte(...);
...
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); } ... }
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); } ... }
Control-plane API in C

```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);
}

Generated by compiler
```
Deparser: Update the Packet

control Deparser(in Headers hdrs, packet_out packet) {
  apply {
    packet.emit(hdrs.ethernet);
    packet.emit(hdrs.ipv4);
  }
}
Deparser: Update the Packet

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

```c
bpf_xdp_adjust_head(skb, offset);
ebpf_byte = ((char*)(&hd.ethernet.destination))[0];
write_byte(ebpf_packetStart, BYTES(ebpf_packetOffsetInBits) + 0, ebpf_byte);
...
ebpf_packetOffsetInBits += 48;
```
Complete C program structure

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(amount);
    // 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 corresponding action

- **Deparser**
  - Convert headers back into a byte stream.
  - Only valid headers are emitted.
Performance Benchmarks
Performance Evaluation

- **P4C-XDP binary**
  - `./p4c-xdp --target xdp -o <output_file> <input_p4>
  - Sample code at tests/xdp*.p4
  - Load to driver by: `ip link set dev eth0 xdp obj xdp1.o`

- **Measure packet rate in Mpps**
  - Packet drop rate (XDP_DROP) and transmit rate (XDP_TX)
Sample P4 Program Performance

- **SimpleDrop**: return XDP_DROP
- **xdp1.p4**: parse Ethernet/IPv4 header, deparse it, and drop.
- **xdp3.p4**: parse Ethernet/IPv4 header, lookup a MAC address in a map, deparse it, and drop.
- **xdp6.p4**: parse Ethernet/IPv4 header, lookup and get a new TTL value from eBPF map, set to IPv4 header, deparse it, and drop.
- **Possible Optimization**: avoid byte-order translation and unnecessary (de-)parsing

<table>
<thead>
<tr>
<th>P4 Program</th>
<th>Performance (Mpps)</th>
<th>Possible Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleDrop</td>
<td>14.4</td>
<td>NA</td>
</tr>
<tr>
<td>xdp1</td>
<td>8.1</td>
<td>14</td>
</tr>
<tr>
<td>xdp3</td>
<td>7.1</td>
<td>13</td>
</tr>
<tr>
<td>xdp6</td>
<td>2.5</td>
<td>12</td>
</tr>
</tbody>
</table>
Limitations
## Fundamental Limitations

<table>
<thead>
<tr>
<th>Feature</th>
<th>P4</th>
<th>XDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loops</td>
<td>Parsers</td>
<td>Tail call</td>
</tr>
<tr>
<td>Nested headers</td>
<td>Bounded depth</td>
<td>Bounded depth</td>
</tr>
<tr>
<td>Multicast/broadcast</td>
<td>External</td>
<td>No</td>
</tr>
<tr>
<td>Packet segmentation</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Packet reassembly</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Timers/timeouts/aging</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Queues</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Scheduling</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>State</td>
<td>Registers/counters</td>
<td>Maps</td>
</tr>
<tr>
<td>Linear scans</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Limitations of XDP

• No multi-/broadcast support
  • No ability to clone packets in XDP

• The stack size is too small
  • Complex pipelines are rejected by the verifier

• Generic XDP and TCP
  • TCP is ignored by the generic XDP driver

• eBPF maps cannot be pinned in network namespaces
Conclusion

• P4 is a language that defines data-path behavior
  • It generalizes to different architectures
  • Including the Linux kernel

• P4 can express XDP
  • High-level abstraction to C code
  • Generated code is performant but not optimal
  • Many future optimizations are possible

• P4 and XDP have similar limitations