
*

OfP4, a P4 front-end for Open vSwitch

Ben Pfaff
Debnil Sur
Leonid Ryzhyk
Mihai Budiu

[*] Image generated by Stable Diffusion with the title of this talk as a prompt.

vmware®

What is P4?

as compared to OpenFlow

Similarities to OpenFlow

- Flow tables

Differences from OpenFlow

- Hardware centric
- Target specific
- Compiled (p4c)
- Typed flow tables
- A real language
  - Control flow
  - Expressions
  - Arithmetic

Other points

- Not a dead standard
- Increasing availability
- P4Runtime control protocol

Existing P4 Software Switches

(as of about March 2022)

Others in development:

- uBPF
- DPDK
- PSA eBPF

BMv2

\+ Accurate simulation
\- Low performance.

T_4P_4S

\+ Fast
\- Hard to install across operating systems.

PISCES

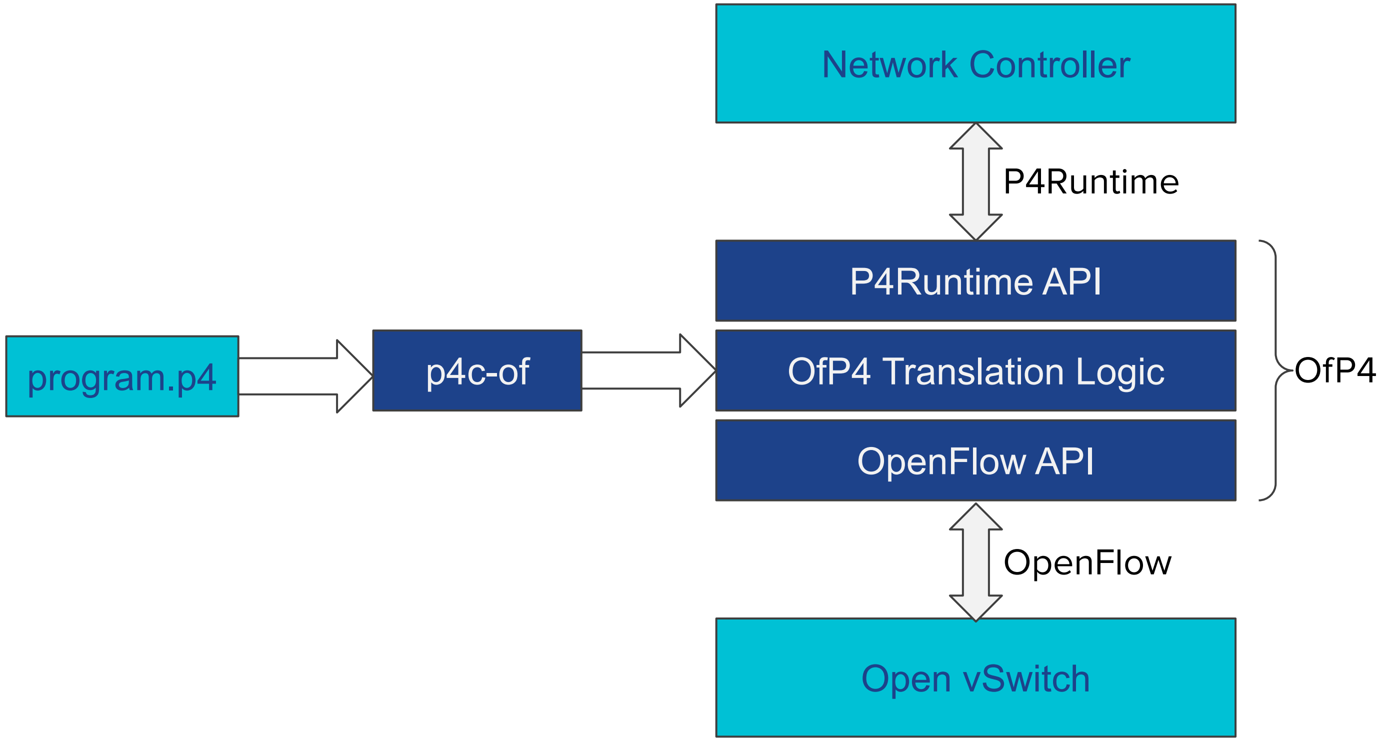
\- Unmaintained
\- No P4Runtime support

P4 for OVS

\- Hardware focus (?)
\- Uncertain timeline

OfP4: Software P4 with an OVS data plane

A daemon to translate between P4+P4Runtime and OpenFlow



Unmodified, upstream OVS
Uses OVS extensions to OpenFlow

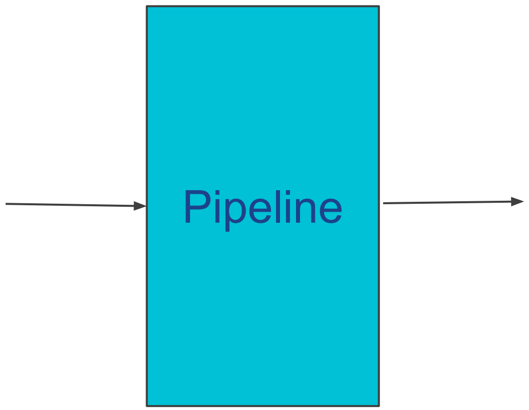
Starting from a P4 program and the controller that supports it:

- Compile P4 with p4c-of
- Connect controller to OfP4 over P4Runtime
- Connect OfP4 to Open vSwitch over OpenFlow

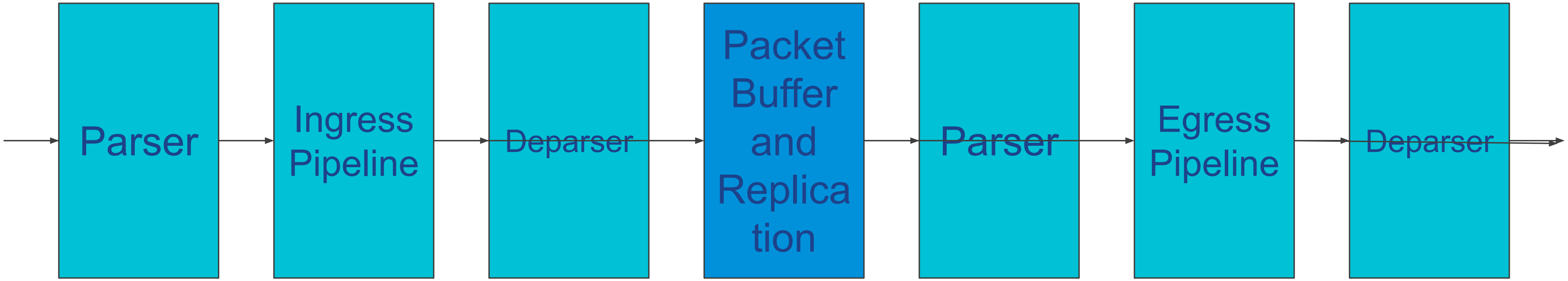
vmware

A P4 Architecture for OVS

OpenFlow Architecture



P4 Portable Switch Architecture



Fields in P4 for common targets

Programmer-Defined Headers

```
header ethernet_t {
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16>   etherType;
}

header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    bit<32>  srcAddr;
    bit<32> dstAddr;
}

struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
}
```

Programmer-Defined Parser

```
parser MyParser(...) {
    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            0x800: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}
```

Fields in P4 for OfP4

~~Programmer-Defined~~ Headers

header ethernet_t {
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16>   etherType;
}

header ipv4_t {
    ~~bit<4>   version;~~
    bit<4>   ihl;
    bit<8>   diffserv;
    ~~bit<16>   totalLen;~~
    ~~bit<16>   identification;~~
    ~~bit<3>   flags;~~
    bit<13>   fragOffset;
    bit<8>   ttl;
    bit<8>   protocol;
    ~~bit<16>   hdrChecksum;~~
    bit<32> srcAddr;
    bit<32> dstAddr;
}

struct headers {
    ethernet_t   ethernet;
    ipv4_t   ipv4;
}

Progr mer-Define arser
p Parser(...)
t {
n pa rnet;
}

state t {
pac r.ethernet);
tran (hdr.ethernet.etherType) {
0x e_ipv4;
de t;
}
}

sta _i
xtra 4);
on acc

}

P4 Metadata

Standard Metadata for Common Targets

```
struct standard_metadata_t {
   bit<9>    ingress_port;
   bit<9>    egress_spec;
   bit<9>    egress_port;
   bit<32>    instance_type;
   bit<32>    packet_length;
   bit<32> enq_timestamp;
   bit<19> enq_qdepth;
   bit<32> deq_timedelta;
   bit<19> deq_qdepth;
   bit<48> ingress_global_timestamp;
   bit<48> egress_global_timestamp;
   bit<16> mcast_grp;
   bit<16> egress_rid;
   bit<1>  checksum_error;
   error parser_error;
   bit<3> priority;
}
```

Program Metadata

```
struct metadata_t {
        bit<32> a;
...
        bit<8>  y;
        bit<8>  z;
}
```

Translating P4 **Metadata** to OpenFlow

P4: Flexible Metadata

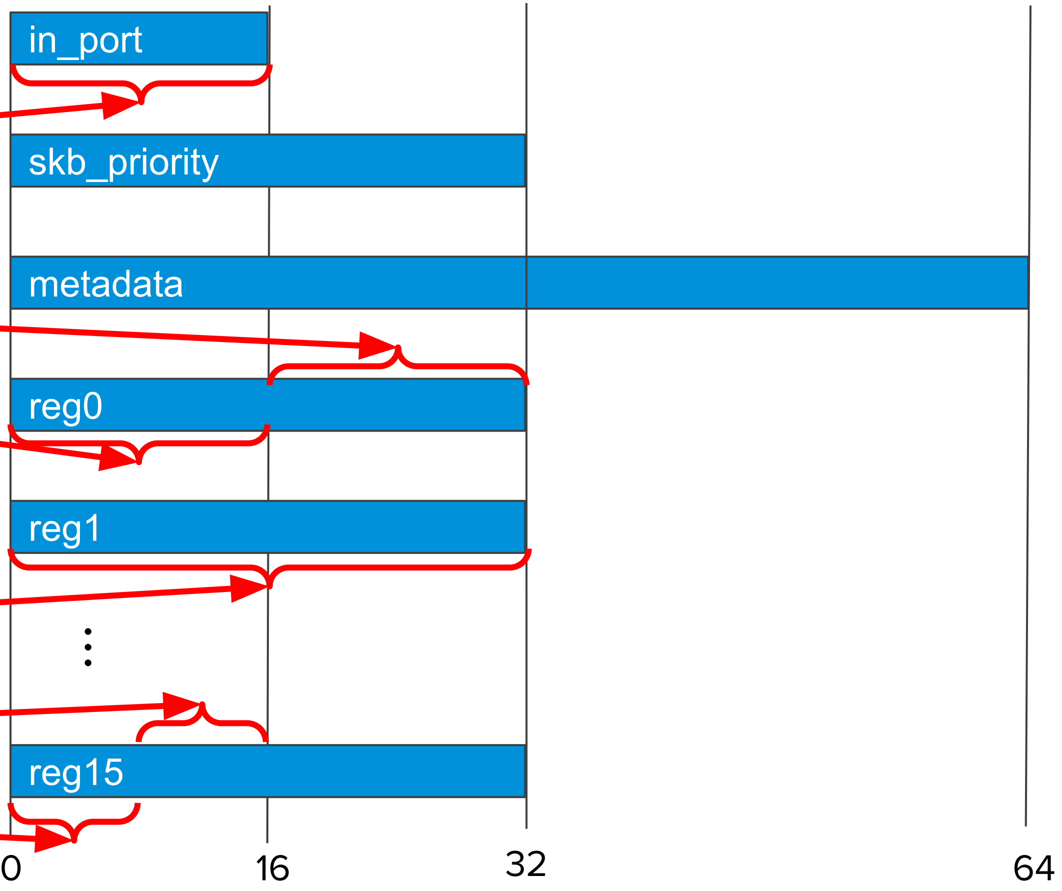
Standard Metadata

```
struct standard_metadata_t {
        bit<16> in_port;
...
        bit<16> out_group;
        bit<16> out_port;
}
```

Program Metadata

```
struct metadata_t {
        bit<32> a;
        ...
        bit<8>  y;
        bit<8>  z;
}
```

OVS: Fixed Metadata



Translating P4 **Table Keys** to OpenFlow

P4: Typed Table Keys

```
table InputVlan {
    key = {
        standard_metadata.in_port: exact;
        hdr.vlan.isValid(): exact;
        hdr.vlan.vid: optional;
    }
    actions = { Drop; SetVlan; UseTaggedVlan; }
    default_action = Drop;
}
```

OpenFlow: Free-Form Matches

in_port=*PORT*

match on false: vlan_tci=0/0x1000

match on true: vlan_tci=0x1000/0x1000

vlan_tci=*VLAN*/0xfff

Translating P4 **Table Actions** to OpenFlow

P4: Typed Actions

```
table InputVlan {
    key = {
        standard_metadata.ingress_port: exact;
        hdr.vlan.isValid(): exact;
        hdr.vlan.vid: optional;
    }
    actions = { Drop; SetVlan; UseTaggedVlan; }
    default_action = Drop;
}
action Drop() {
    mark_to_drop(standard_metadata);
    exit;
}
action SetVlan(bit<16> vid) { meta.vlan = vid; }
action UseTaggedVlan() { meta.vlan = hdr.vlan.vid; }
```

OpenFlow: Free-Form Actions

```
actions=load(0->reg3), resubmit(,31)

actions=load(vid->reg7[0..11]), resubmit(,3)

actions=move(vlan_tci[0..11]->reg7[0..11]), resubmit(,3)
```

Flow Priorities and Default Actions

```
table InputVlan {
        key = {
                standard_metadata.in_port: exact;
                hdr.vlan.isValid(): exact;
                hdr.vlan.vid: optional;
        }
        actions = { Drop; SetVlan; UseTaggedVlan; }
        default_action = Drop;
}
action Drop() {
        mark_to_drop(standard_metadata);
        exit;
}
action SetVlan(bit<16> vid) { meta.vlan = vid; }
action UseTaggedVlan() { meta.vlan = hdr.vlan.vid; }
```

table=2, **priority=0,** actions=load(0->reg3), resubmit(,31)

P4 Control Flow in OpenFlow

An ‘if’ becomes a simple flow table

P4 Control Flow Example

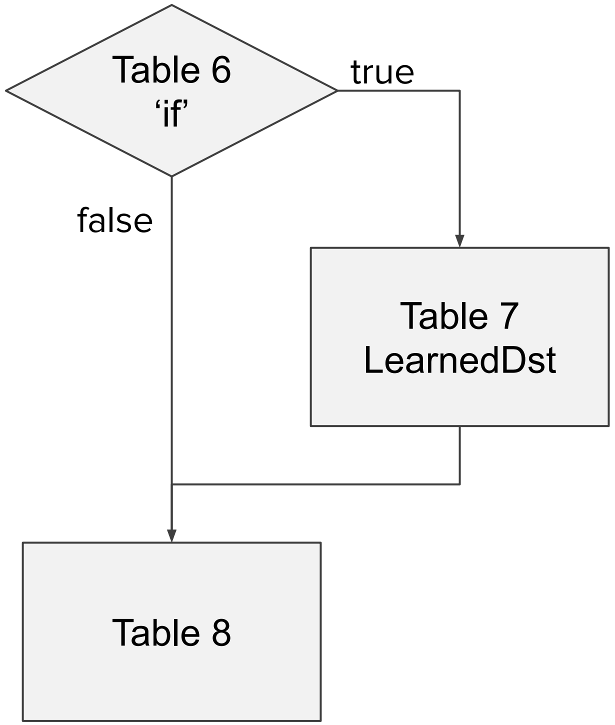
reg5[12] == 0

```
if (!meta.flood && !eth_addr_is_multicast(hdr.eth.dst)) {
        LearnedDst.apply();
}
```

OpenFlow table 7

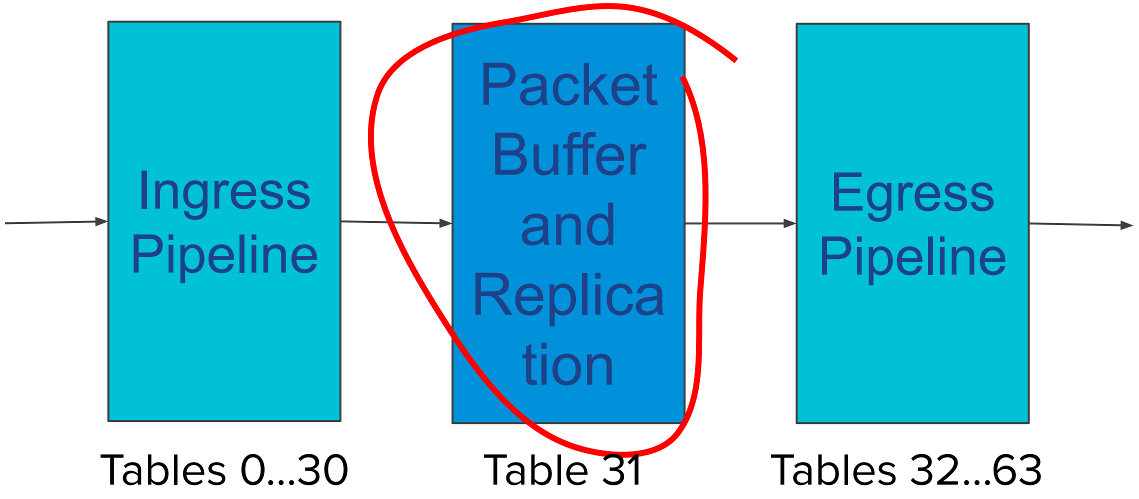
dl_dst[40] == 0

OpenFlow Table Flow



table=6, priority=1, reg5=0/0x1000, dl_dst=0/0x10000000000, actions=resubmit(,7)
table=6, priority=0, actions=resubmit(,8)

Implementing P4 Pipelines



Multicast Group Assignments

Group ID	Ports
12	[1, 2, 3, 4, 5]
23	[1, 2]
34	[3, 4, 5]
…	…

Multicast Flows

table=31, priority=1, reg1=**12**, actions=clone(load: **1-**>reg0, resubmit(,32)),
clone(load:**2**->reg0, resubmit(,32)),
clone(load:**3**->reg0, resubmit(,32)),
clone(load:**4**->reg0, resubmit(,32)),
clone(load:**5**->reg0, resubmit(,32))

table=31, priority=1, reg1=**23**, actions=clone(load:**1-**>reg0, resubmit(,32)),
clone(load:**2**->reg0, resubmit(,32))

Fallback Flow

table=31, priority=**0**, actions=resubmit(,32)

P4 Digests

(not yet implemented)

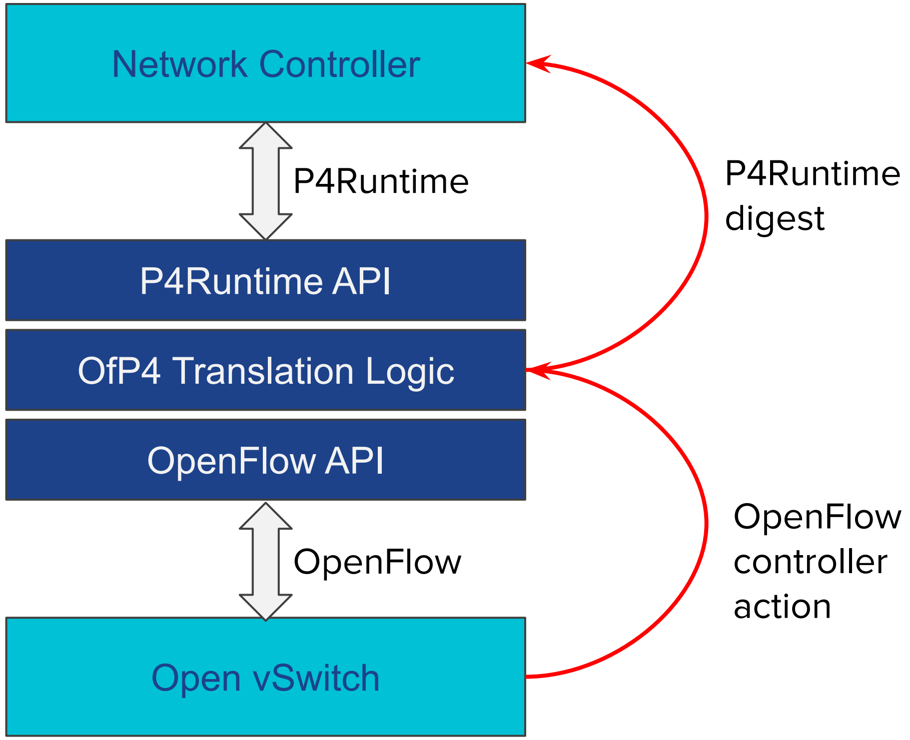
<u>P4 Digest Usage</u>

```
#define ID 0x1234
struct LearnDigest { … }

…

// If the source MAC isn't known, send it
// to the control plane to be learned.
if (!meta.flood
    && !eth_addr_is_multicast(hdr.eth.src)
    && !LearnedSrc.apply().hit) {
        LearnDigest d;
        d.port = meta_in.in_port;
        d.vlan = meta.vlan;
        d.mac = hdr.eth.src;
        digest<LearnDigest>(ID, d);
}
```

actions=controller(userdata=1234)



Limitations

- Arithmetic
- Table number limits
- Metadata size limits
- Compatibility
- Brittleness

 *

Thank You

https://github.com/vmware/nerpa

[*] Image generated by Stable Diffusion with the title of this talk as a prompt.