P4: Protocol-Independent Packet Processors

Guest Lecture ECEN5013, September 29th, 2015

Oliver Michel

Next Generation Networks Research Group

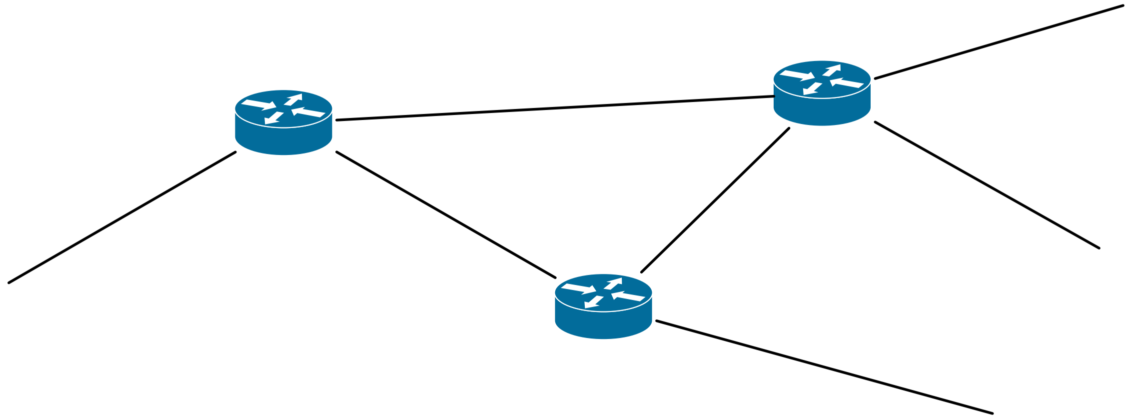
University of Colorado Boulder

Outline

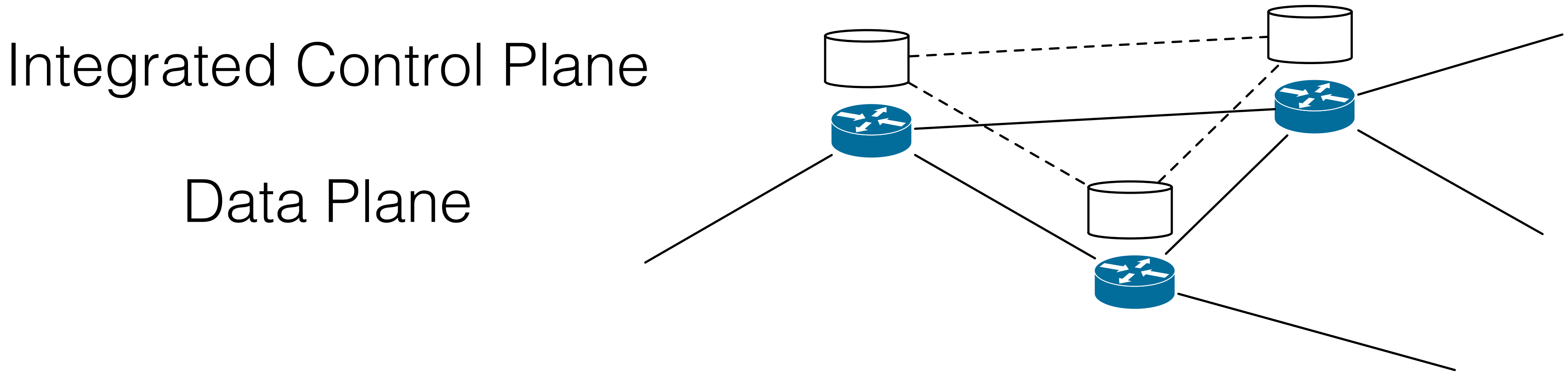
1. SDN/Open Flow
2. Open Flow Limitations
3. Protocol-Independent Processing
4. Abstract Forwarding Model and the P4 Language
5. Demo
6. Conclusion

SDN in one slide

Data Plane

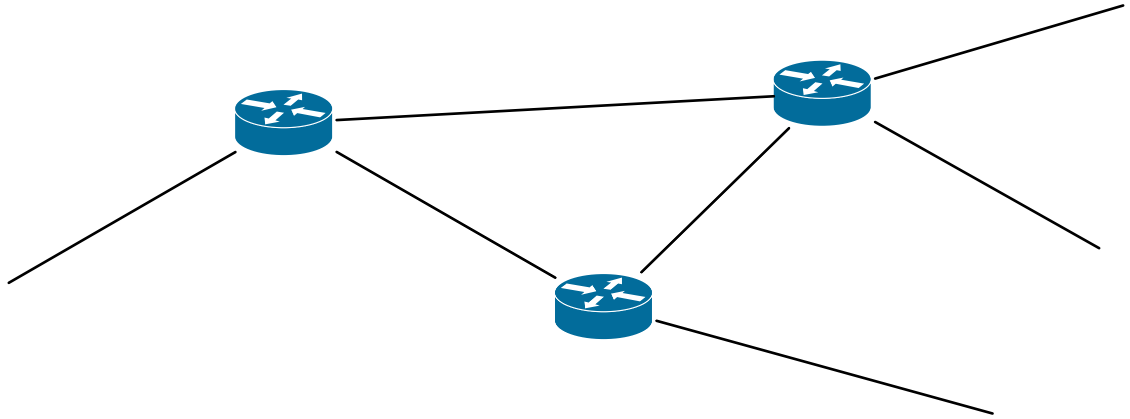


SDN in one slide



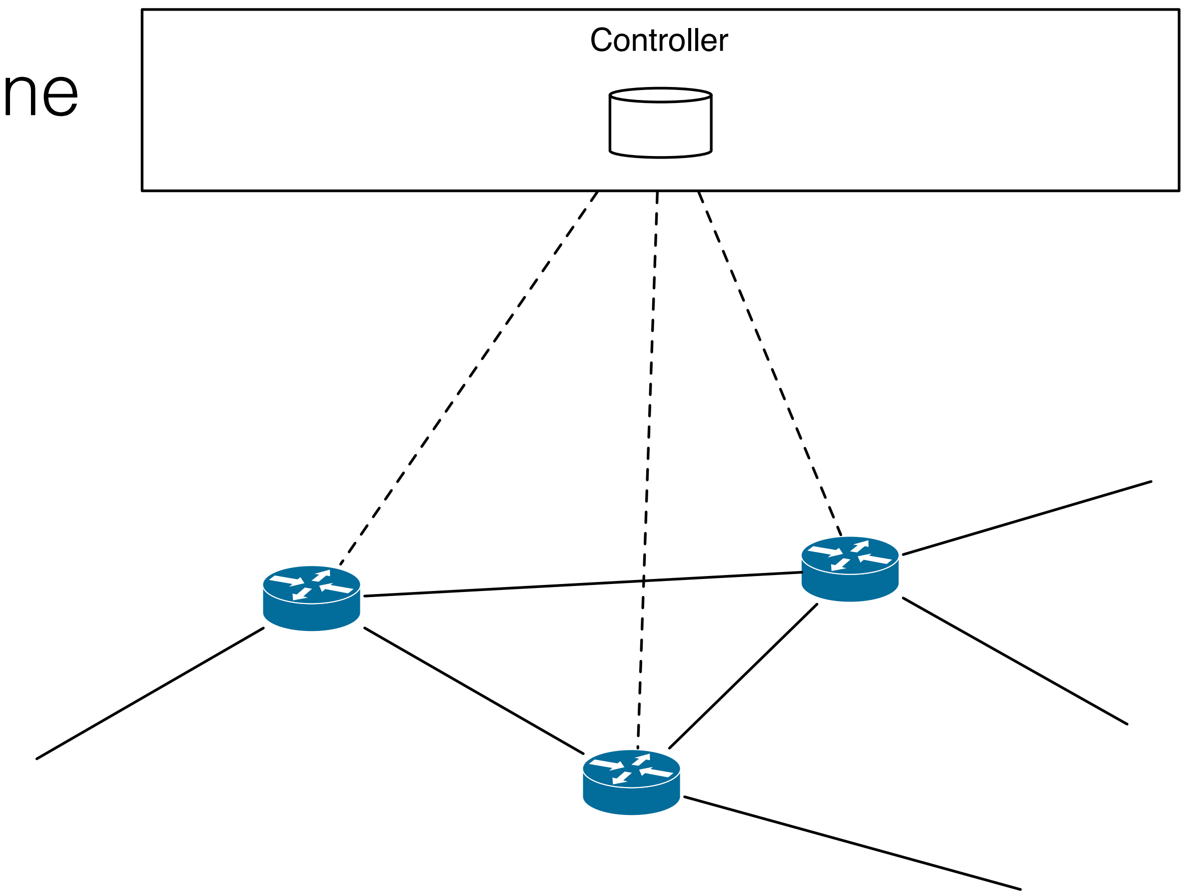
SDN in one slide

Data Plane



SDN in one slide

Decoupled Control Plane



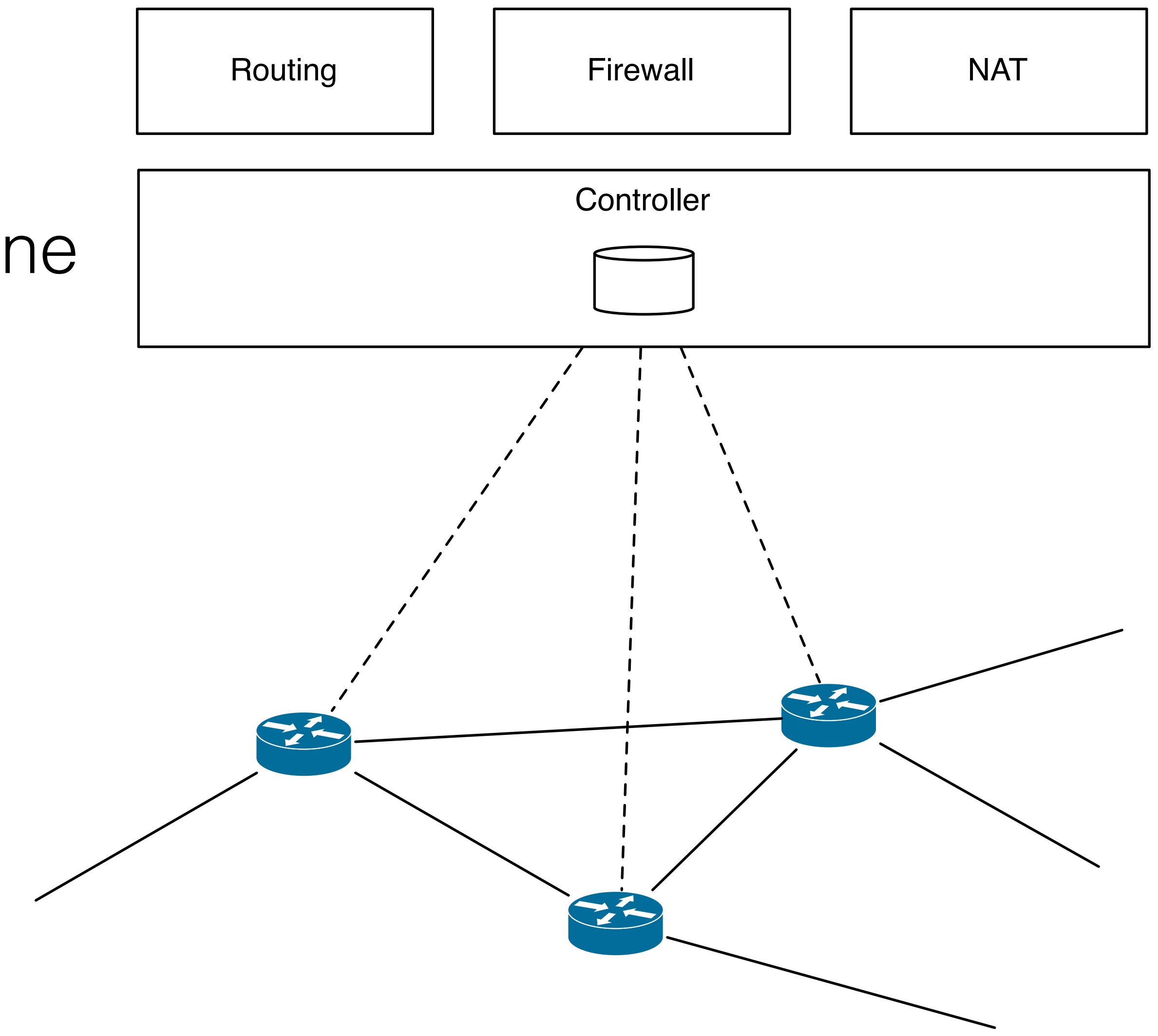
Data Plane

SDN in one slide

SDN Applications

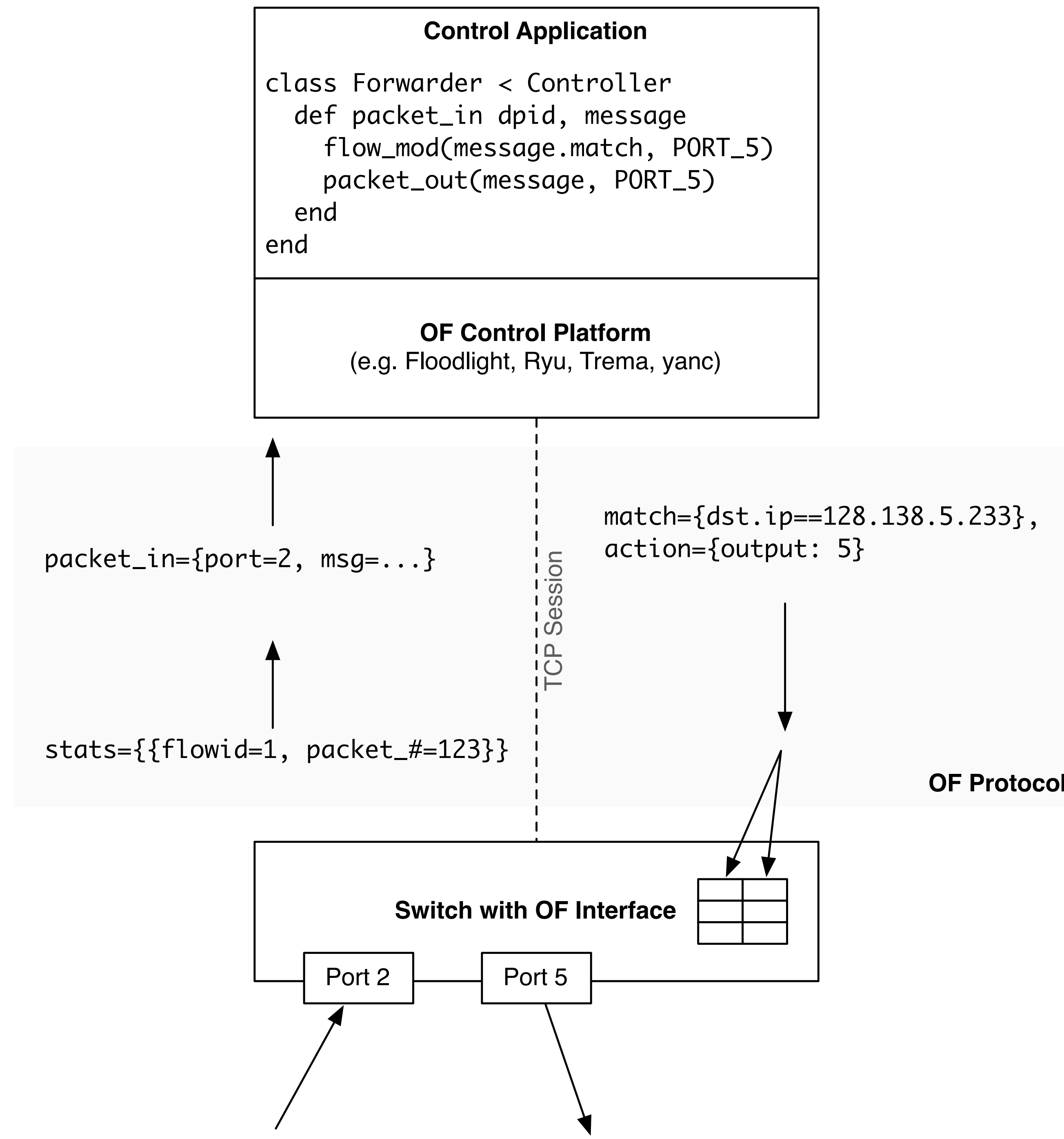
Decoupled Control Plane

Data Plane



OpenFlow in one slide

- open protocol that gives applications control over a switches data plane
- designed around a set of header match fields and forwarding actions



OpenFlow Match/Action

- TCAM Model
- OF Wire Protocol 1.4 (Oct 2013): 41 match header fields
- Most H/W switches only support limited match/action set (Ethernet, IP, (TCP, MPLS)) due to ASIC limitations

[Open Networking Foundation. Open Flow Switch Specification 1.4.0]

```
enum oxm_ofb_match_fields {
  OFPXMT_OFB_IN_PORT,
  OFPXMT_OFB_IN_PHY_PORT,
  OFPXMT_OFB_METADATA,
  OFPXMT_OFB_ETH_DST ,
  OFPXMT_OFB_ETH_SRC ,
  OFPXMT_OFB_ETH_TYPE ,
  OFPXMT_OFB_VLAN_VID ,
  OFPXMT_OFB_VLAN_PCP ,
  OFPXMT_OFB_IP_DSCP ,
  OFPXMT_OFB_IP_ECN ,
  OFPXMT_OFB_IP_PROTO ,
  OFPXMT_OFB_IPV4_SRC ,
  OFPXMT_OFB_IPV4_DST ,
  OFPXMT_OFB_TCP_SRC ,
  OFPXMT_OFB_TCP_DST ,
  OFPXMT_OFB_UDP_SRC ,
  OFPXMT_OFB_UDP_DST ,
  OFPXMT_OFB_SCTP_SRC ,
  OFPXMT_OFB_SCTP_DST ,
  OFPXMT_OFB_ICMPV4_TYPE ,
  OFPXMT_OFB_ICMPV4_CODE ,
  OFPXMT_OFB_ARP_OP ,
  OFPXMT_OFB_ARP_SPA ,
  OFPXMT_OFB_ARP_TPA ,
  OFPXMT_OFB_ARP_SHA ,
  OFPXMT_OFB_ARP_THA ,
  OFPXMT_OFB_IPV6_SRC ,
  OFPXMT_OFB_IPV6_DST ,
  OFPXMT_OFB_IPV6_FLABEL ,
  OFPXMT_OFB_ICMPV6_TYPE ,
  OFPXMT_OFB_ICMPV6_CODE ,
  OFPXMT_OFB_IPV6_ND_TARGET,
  OFPXMT_OFB_IPV6_ND_SLL,
  OFPXMT_OFB_IPV6_ND_TLL,
  OFPXMT_OFB_MPLS_LABEL,
  OFPXMT_OFB_MPLS_TC,
  OFPXMT_OFP_MPLS_BOS,
  OFPXMT_OFB_PBB_ISID,
  OFPXMT_OFB_TUNNEL_ID,
  OFPXMT_OFB_IPV6_EXTHDR,
  OFPXMT_OFB_PBB_UCA
};
```

```
enum ofp_action_type {
  OFPAT_OUTPUT,
  OFPAT_COPY_TTL_OUT,
  OFPAT_COPY_TTL_IN,
  OFPAT_SET_MPLS_TTL,
  OFPAT_DEC_MPLS_TTL,
  OFPAT_PUSH_VLAN,
  OFPAT_POP_VLAN,
  OFPAT_PUSH_MPLS,
  OFPAT_POP_MPLS,
  OFPAT_SET_QUEUE,
  OFPAT_GROUP,
  OFPAT_SET_NW_TTL,
  OFPAT_DEC_NW_TTL,
  OFPAT_SET_FIELD,
  OFPAT_PUSH_PBB,
  OFPAT_POP_PBB,
  OFPAT_EXPERIMENTER
};
```

Open Flow is a balancing act

- forwarding abstraction balancing…

  1. general match/action (TCAM model)

  2. fixed-function switch ASICs (often only 12 fixed fields)

- Why?

  - long development cycles and major cost require very clear long-time guidelines

Enabling Innovation?

ARP ICMP UDP SCTP RSVP
IP ECN IGMP L2TP PPP DNS
Ethernet BGP DHCP HTTP
SNMP IPsec TLS NNTP POP

OpenFlow Original Paper
[SIGCOMM CCR 38/2]

OpenFlow: Enabling Innovation in Campus Networks

March 14, 2008

Nick McKeown, Stanford University
Tom Anderson, University of Washington
Hari Balakrishnan, MIT
Guru Parulkar, Stanford University
Larry Peterson, Princeton University
Jennifer Rexford, Princeton University
Scott Shenker, University of California, Berkeley
Jonathan Turner, Washington University in St. Louis

ABSTRACT

This whitepaper proposes OpenFlow: a way for researchers to run experimental protocols in the networks they use every day. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our goal is to encourage networking vendors to add OpenFlow to their switch products for deployment in college campus backbones and wiring closets. We believe that OpenFlow is a pragmatic compromise: on one hand, it allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate and with high port-density; while on the other hand, vendors do not need to expose the internal workings of their switches. In addition to allowing researchers to evaluate their ideas in real-world traffic settings, OpenFlow could serve as a useful campus component in proposed large-scale testbeds like GENI. Two buildings at Stanford University will soon run OpenFlow networks, using commercial Ethernet switches and routers. We will work to encourage deployment at other schools; and We encourage you to consider deploying OpenFlow in your university network too.

Categories and Subject Descriptors

C.2 [**Internetworking**]: Routers

General Terms

Experimentation, Design

Keywords

Ethernet switch, virtualization, flow-based

1. THE NEED FOR PROGRAMMABLE NETWORKS

Networks have become part of the critical infrastructure of our businesses, homes and schools. This success has been both a blessing and a curse for networking researchers; their work is more relevant, but their chance of making an impact is more remote. The reduction in real-world impact of any given network innovation is because the enormous installed base of equipment and protocols, and the reluctance to experiment with production traffic, which have created an exceedingly high barrier to entry for new ideas. Today, there is almost no practical way to experiment with new network protocols (e.g., new routing protocols, or alternatives to IP) in sufficiently realistic settings (e.g., at scale carrying real traffic) to gain the confidence needed for their widespread deployment. The result is that most new ideas from the networking research community go untried and untested; hence the commonly held belief that the network infrastructure has "ossified".

Having recognized the problem, the networking community is hard at work developing programmable networks, such as GENI [1] a proposed nationwide research facility for experimenting with new network architectures and distributed systems. These programmable networks call for programmable switches and routers that (using *virtualization*) can process packets for multiple isolated experimental networks simultaneously. For example, in GENI it is envisaged that a researcher will be allocated a *slice* of resources across the whole network, consisting of a portion of network links, packet processing elements (e.g. routers) and end-hosts; researchers program their slices to behave as they wish. A slice could extend across the backbone, into access networks, into college campuses, industrial research labs, and include wiring closets, wireless networks, and sensor networks.

Virtualized programmable networks could lower the barrier to entry for new ideas, increasing the rate of innovation in the network infrastructure. But the plans for nationwide facilities are ambitious (and costly), and it will take years for them to be deployed.

This whitepaper focuses on a shorter-term question closer to home: *As researchers, how can we run experiments in our campus networks?* If we can figure out how, we can start soon and extend the technique to other campuses to benefit the whole community.

To meet this challenge, several questions need answering, including: In the early days, how will college network administrators get comfortable putting experimental equipment (switches, routers, access points, etc.) into their network? How will researchers control a portion of their local network in a way that does not disrupt others who depend on it? And exactly what functionality is needed in network switches to enable experiments? Our goal here is to propose a new switch feature that can help extend programmability into the wiring closet of college campuses.

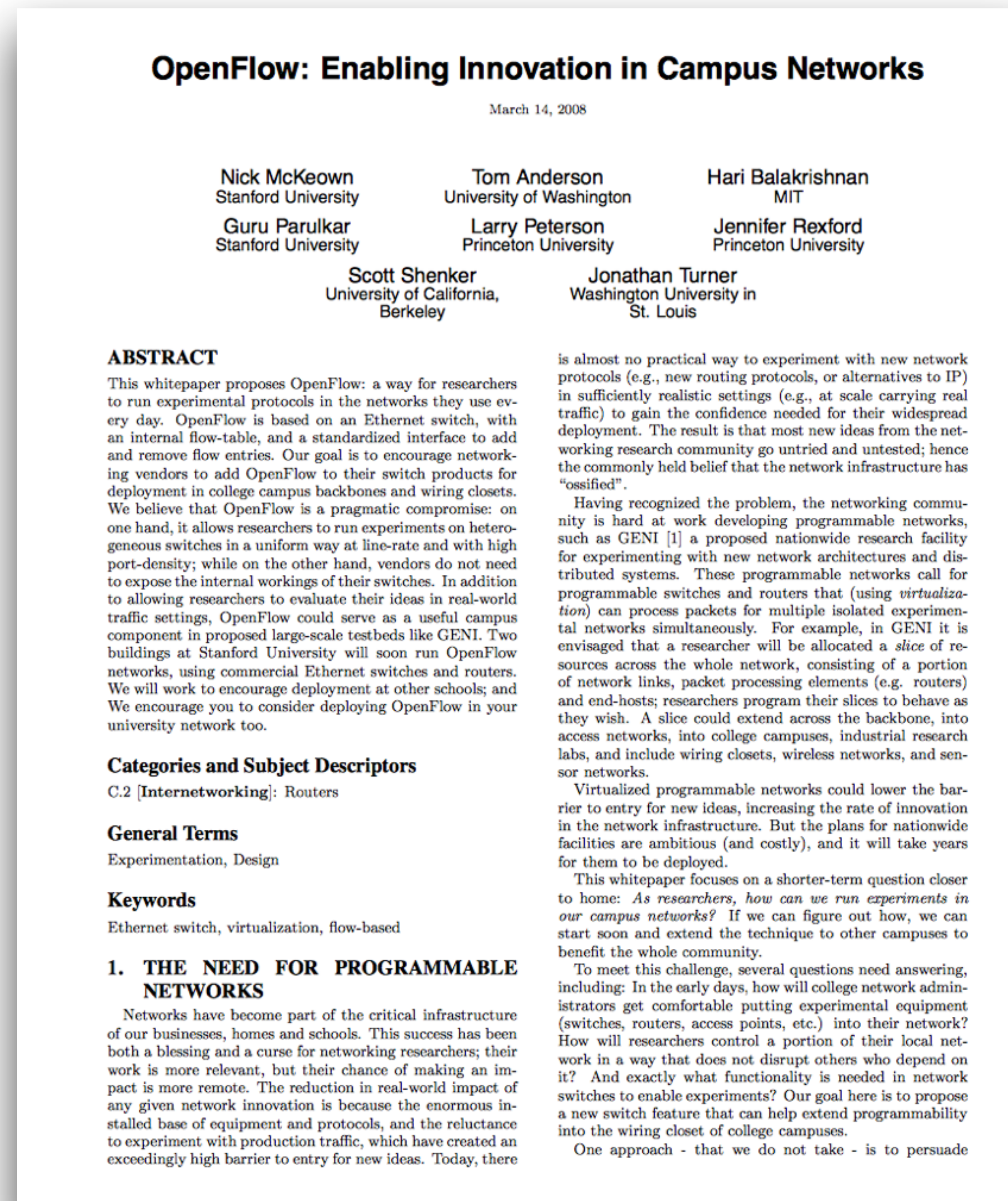
One approach - that we do not take - is to persuade

Enabling Innovation?

ARP ICMP UDP SCTP RSVP
IP ECN IGMP L2TP PPP DNS
Ethernet BGP DHCP HTTP
SNMP IPsec TLS NNTP POP

- limited to existing headers/header fields
- no support for custom (encapsulating) protocols
- NVGRE, VXLAN, STT

OpenFlow Original Paper
[SIGCOMM CCR 38/2]



OpenFlow: Enabling Innovation in Campus Networks

March 14, 2008

Nick McKeown, Stanford University
Tom Anderson, University of Washington
Hari Balakrishnan, MIT
Guru Parulkar, Stanford University
Larry Peterson, Princeton University
Jennifer Rexford, Princeton University
Scott Shenker, University of California, Berkeley
Jonathan Turner, Washington University in St. Louis

ABSTRACT

This whitepaper proposes OpenFlow: a way for researchers to run experimental protocols in the networks they use every day. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our goal is to encourage networking vendors to add OpenFlow to their switch products for deployment in college campus backbones and wiring closets. We believe that OpenFlow is a pragmatic compromise: on one hand, it allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate and with high port-density; while on the other hand, vendors do not need to expose the internal workings of their switches. In addition to allowing researchers to evaluate their ideas in real-world traffic settings, OpenFlow could serve as a useful campus component in proposed large-scale testbeds like GENI. Two buildings at Stanford University will soon run OpenFlow networks, using commercial Ethernet switches and routers. We will work to encourage deployment at other schools; and We encourage you to consider deploying OpenFlow in your university network too.

Categories and Subject Descriptors

C.2 [**Internetworking**]: Routers

General Terms

Experimentation, Design

Keywords

Ethernet switch, virtualization, flow-based

1. THE NEED FOR PROGRAMMABLE NETWORKS

Networks have become part of the critical infrastructure of our businesses, homes and schools. This success has been both a blessing and a curse for networking researchers; their work is more relevant, but their chance of making an impact is more remote. The reduction in real-world impact of any given network innovation is because the enormous installed base of equipment and protocols, and the reluctance to experiment with production traffic, which have created an exceedingly high barrier to entry for new ideas. Today, there is almost no practical way to experiment with new network protocols (e.g., new routing protocols, or alternatives to IP) in sufficiently realistic settings (e.g., at scale carrying real traffic) to gain the confidence needed for their widespread deployment. The result is that most new ideas from the networking research community go untried and untested; hence the commonly held belief that the network infrastructure has "ossified".

Having recognized the problem, the networking community is hard at work developing programmable networks, such as GENI [1] a proposed nationwide research facility for experimenting with new network architectures and distributed systems. These programmable networks call for programmable switches and routers that (using *virtualization*) can process packets for multiple isolated experimental networks simultaneously. For example, in GENI it is envisaged that a researcher will be allocated a *slice* of resources across the whole network, consisting of a portion of network links, packet processing elements (e.g. routers) and end-hosts; researchers program their slices to behave as they wish. A slice could extend across the backbone, into access networks, into college campuses, industrial research labs, and include wiring closets, wireless networks, and sensor networks.

Virtualized programmable networks could lower the barrier to entry for new ideas, increasing the rate of innovation in the network infrastructure. But the plans for nationwide facilities are ambitious (and costly), and it will take years for them to be deployed.

This whitepaper focuses on a shorter-term question closer to home: *As researchers, how can we run experiments in our campus networks?* If we can figure out how, we can start soon and extend the technique to other campuses to benefit the whole community.

To meet this challenge, several questions need answering, including: In the early days, how will college network administrators get comfortable putting experimental equipment (switches, routers, access points, etc.) into their network? How will researchers control a portion of their local network in a way that does not disrupt others who depend on it? And exactly what functionality is needed in network switches to enable experiments? Our goal here is to propose a new switch feature that can help extend programmability into the wiring closet of college campuses.

One approach - that we do not take - is to persuade

Idea

- implement flexible mechanisms for parsing packets and matching (arbitrary) headers fields through common interface
- instead of repeatedly extending OF standard

P4 Goals

1. Reconfigurability

2. Protocol-independence

3. Target Independence

But switches still have ASICs?

- Yes, but…
  - new custom ASICs can achieve such flexibility at terabit speeds [Kangaroo INFOCOM ’10, SDN Chip SIGCOMM ’13, Intel FM6000 switch silicon]
  - some switches are more programmable than others:
    - FPGA (Xilinx, Altera, Corsa)
    - NPU (Ezchip, Netronome)
    - CPU (OVS, …)

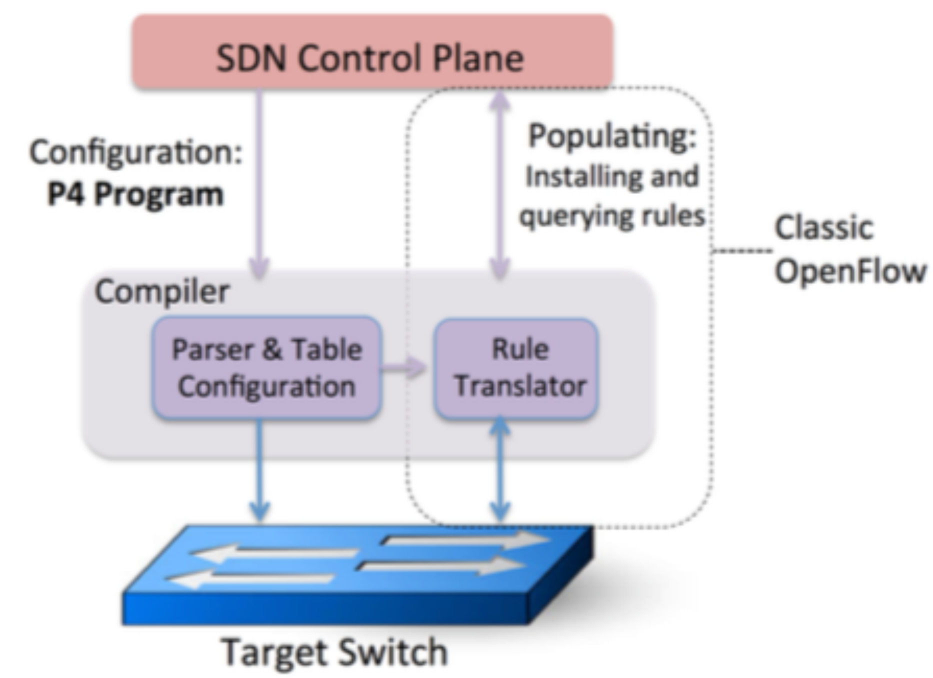
P4 Language

- P4 program configures forwarding behavior (abstract forwarding model)

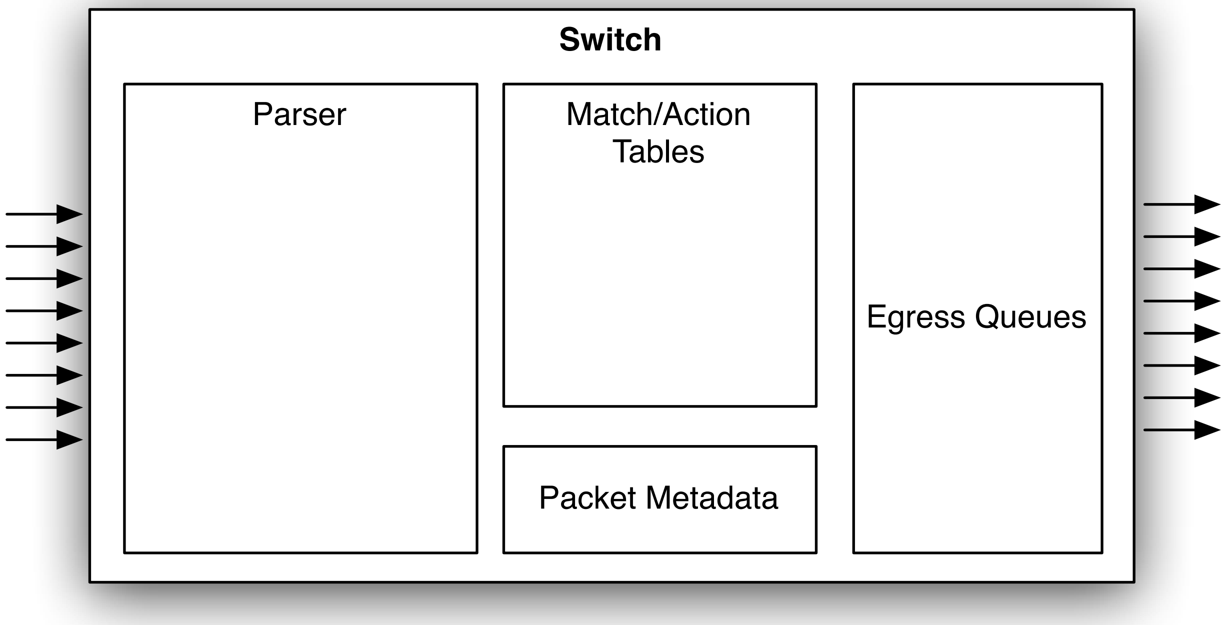
- express serial dependencies (e.g. ARP/L3 Routing)

- P4 compiler translates into a target-specific representation

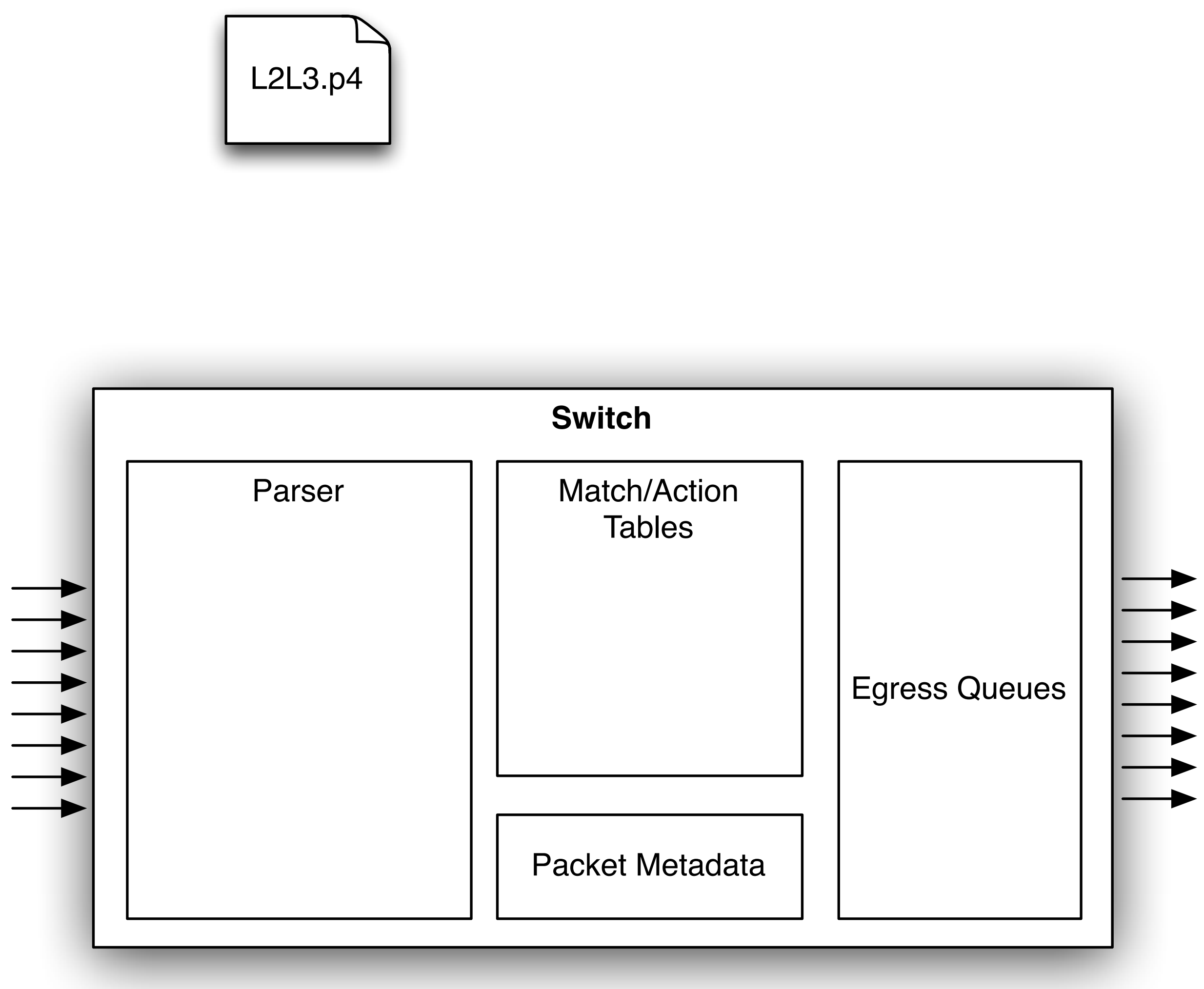
- OF can still be used to install and query rules once forwarding model is defined



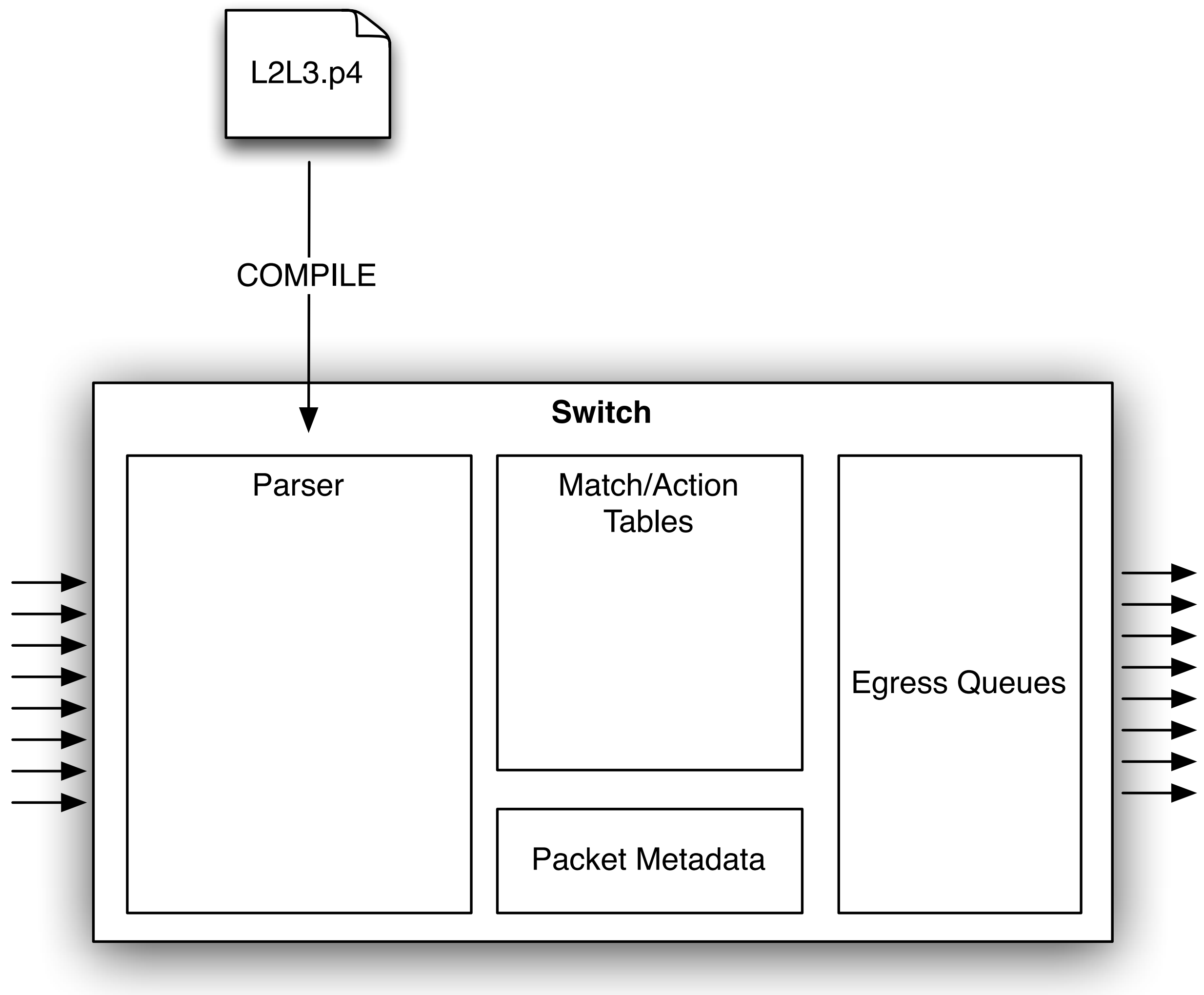
P4 Forwarding Model / Runtime



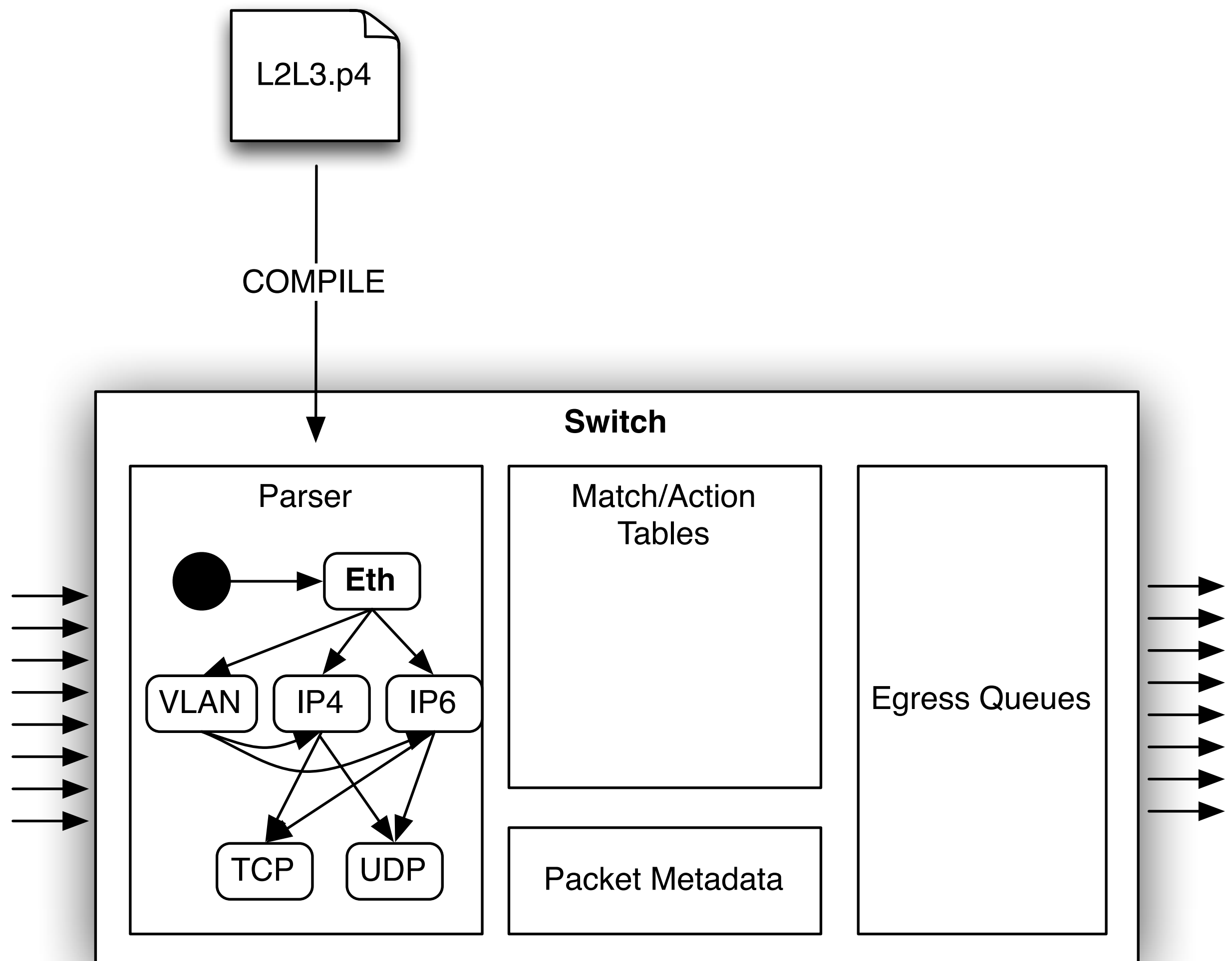
P4 Forwarding Model / Runtime



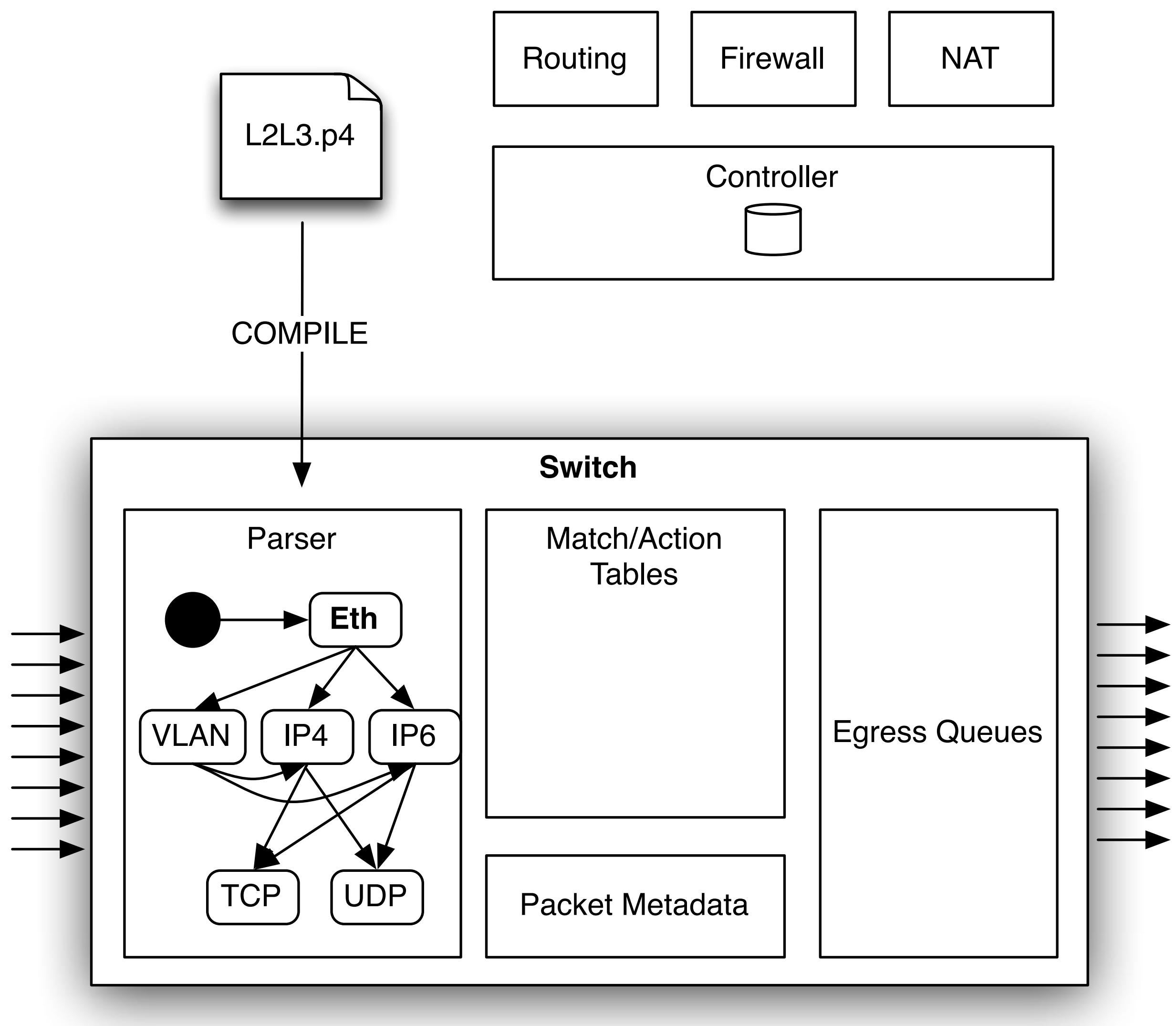
P4 Forwarding Model / Runtime



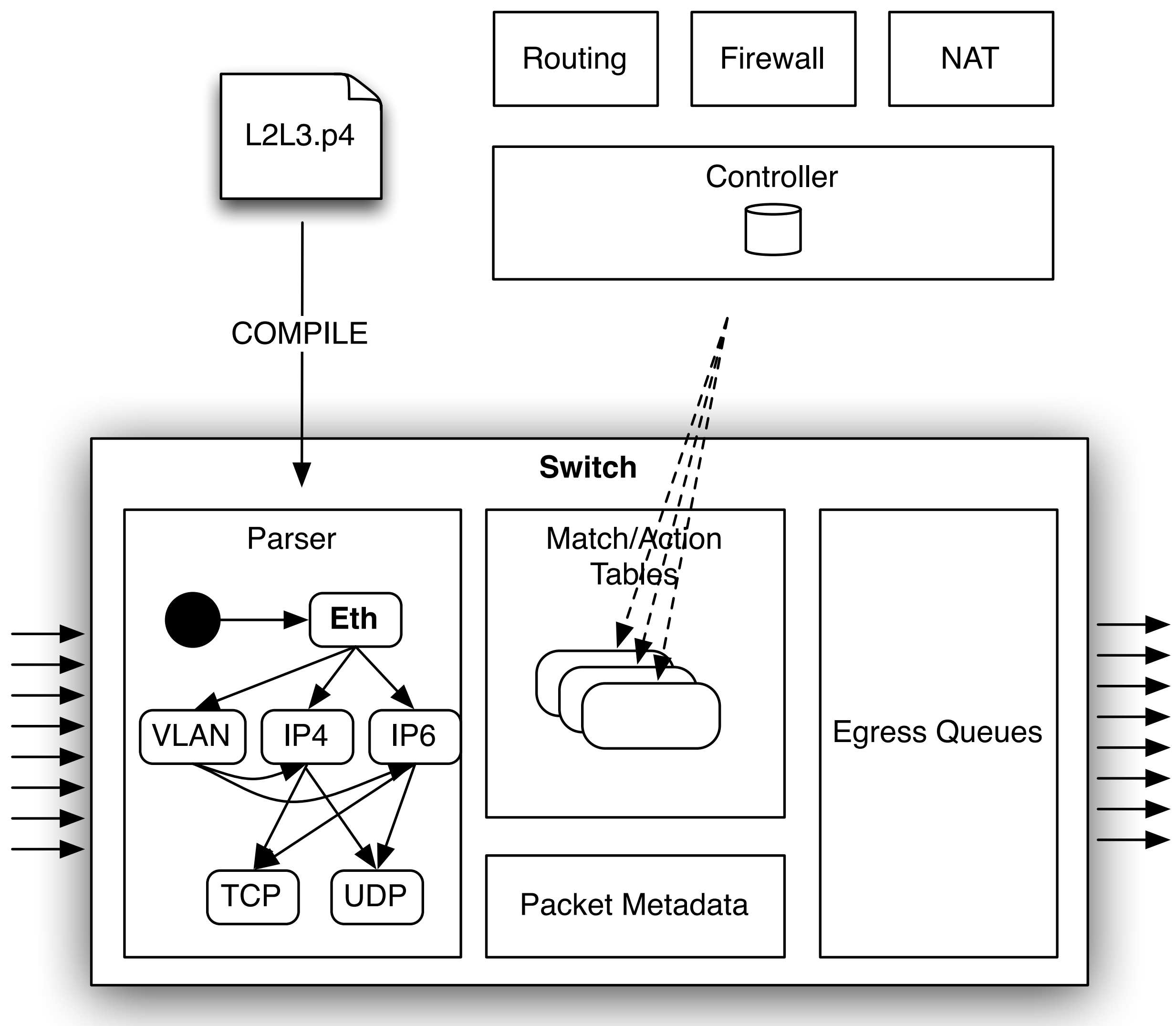
P4 Forwarding Model / Runtime



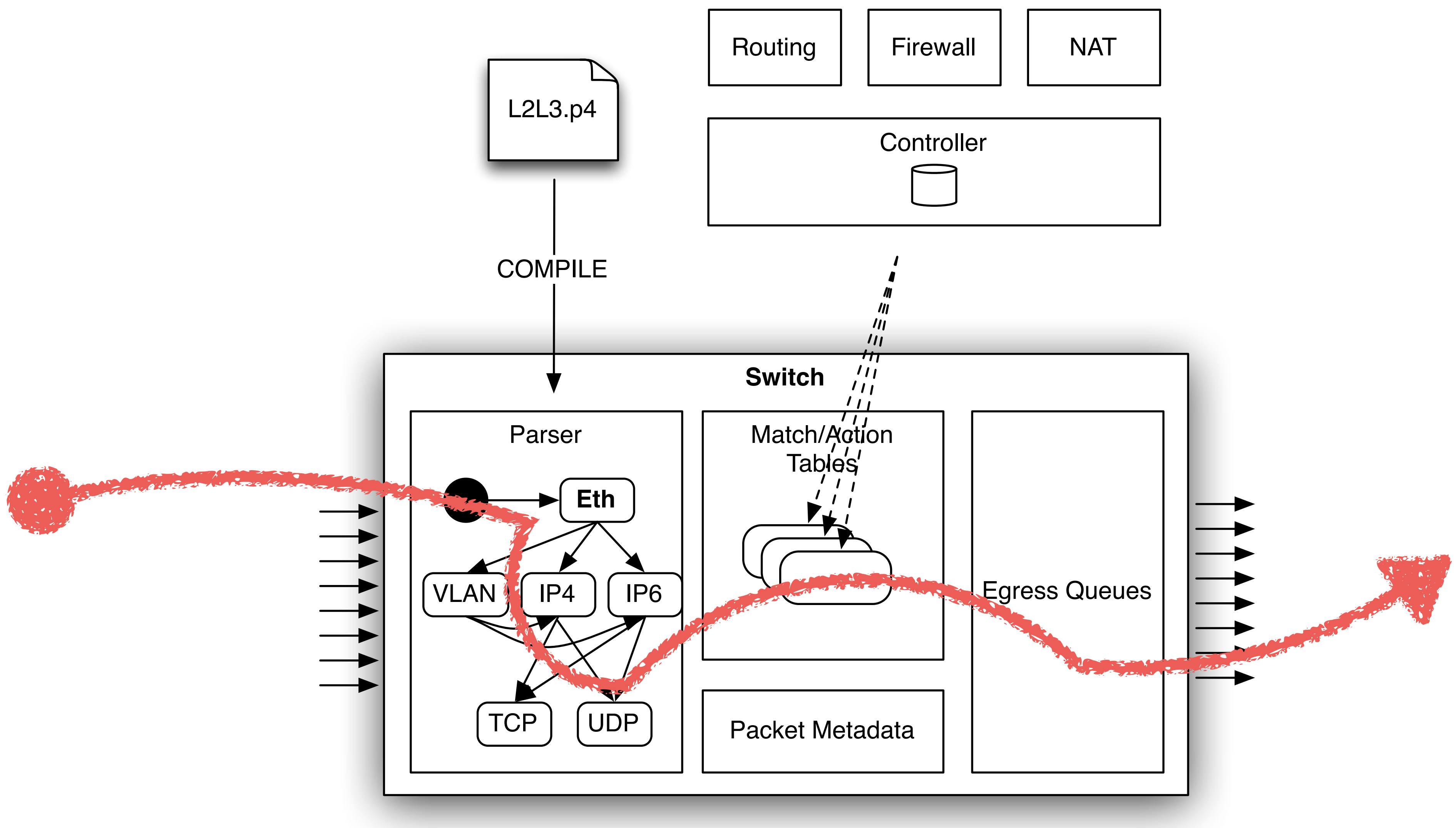
P4 Forwarding Model / Runtime



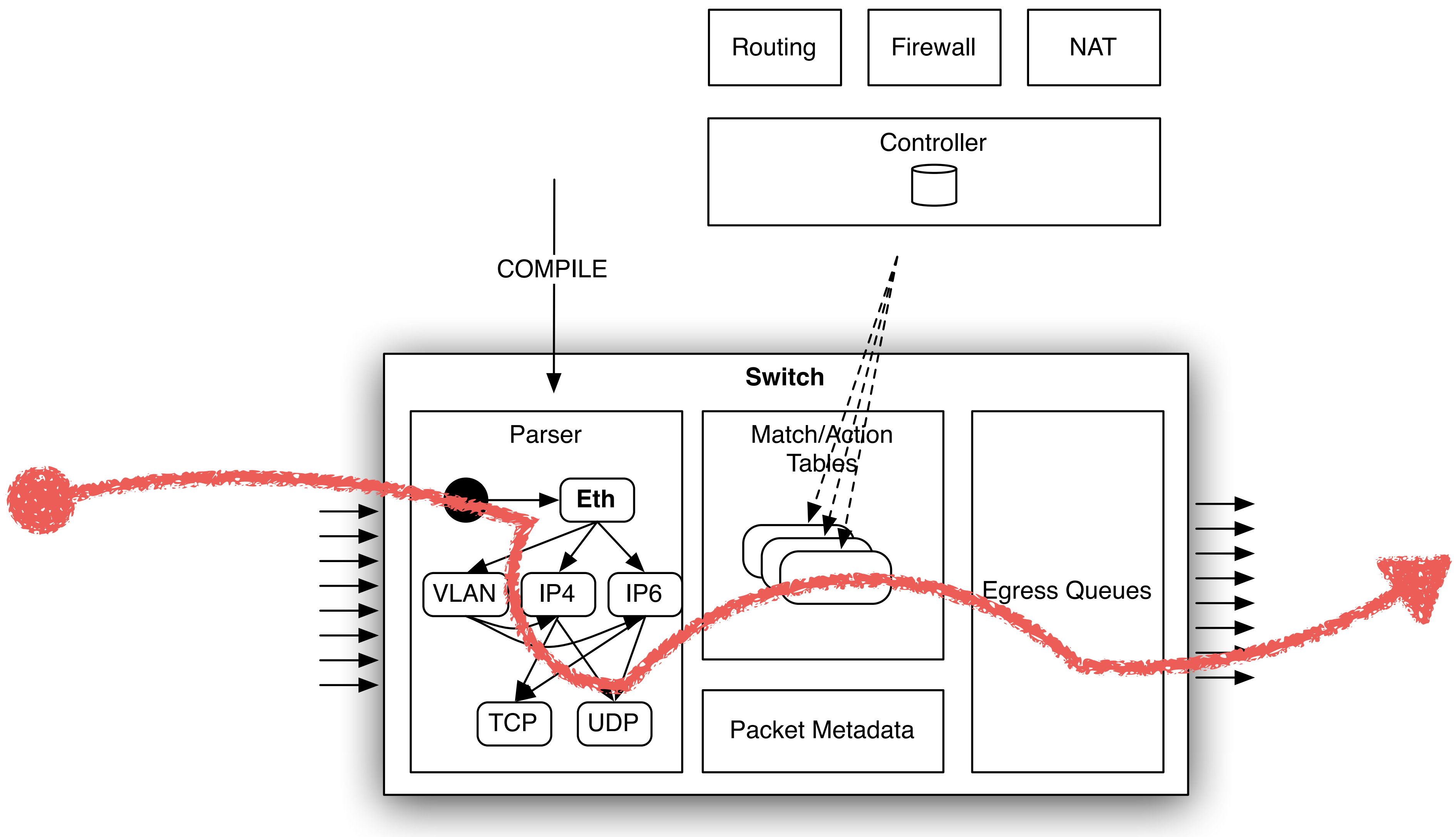
P4 Forwarding Model / Runtime



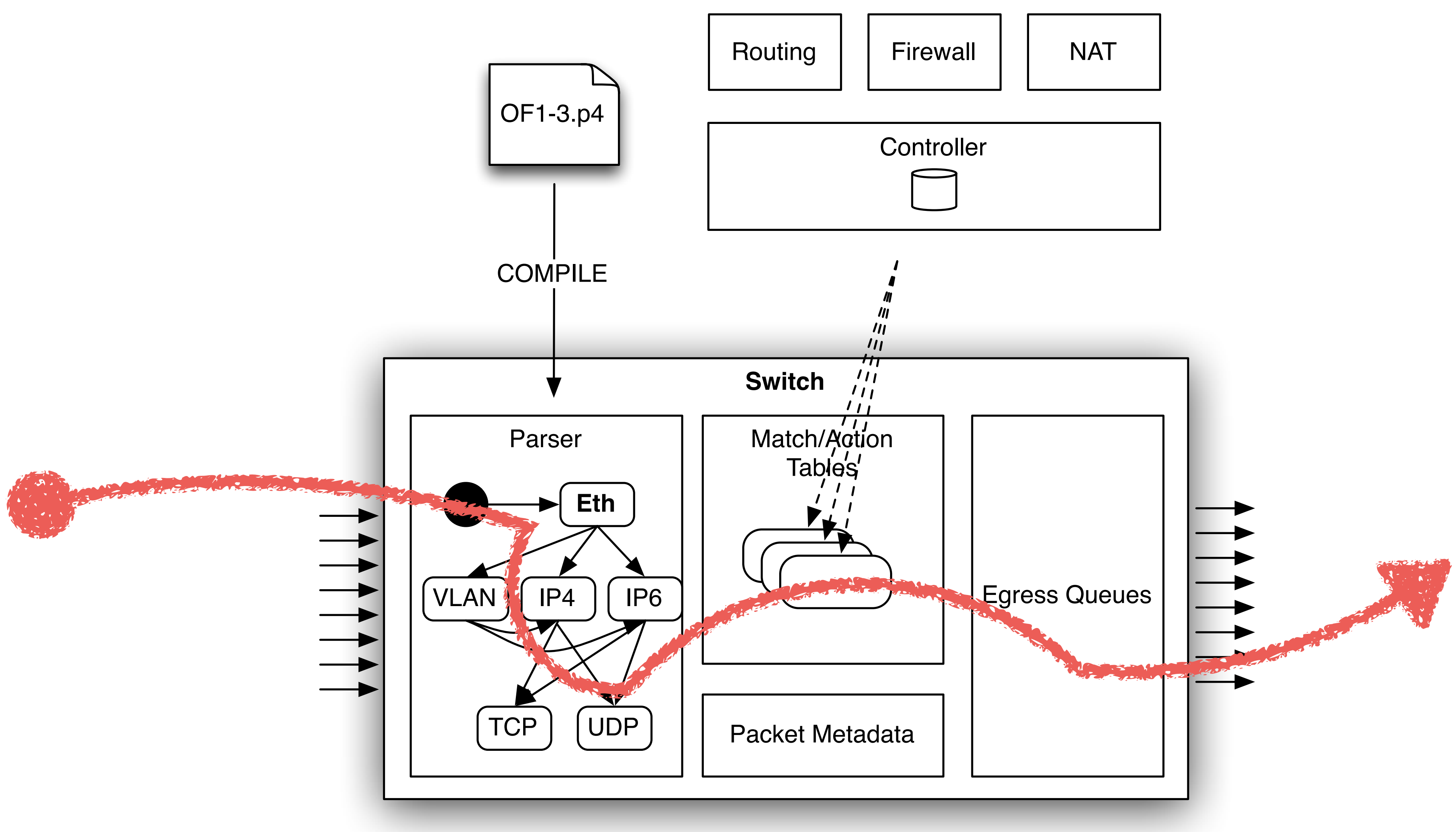
P4 Forwarding Model / Runtime



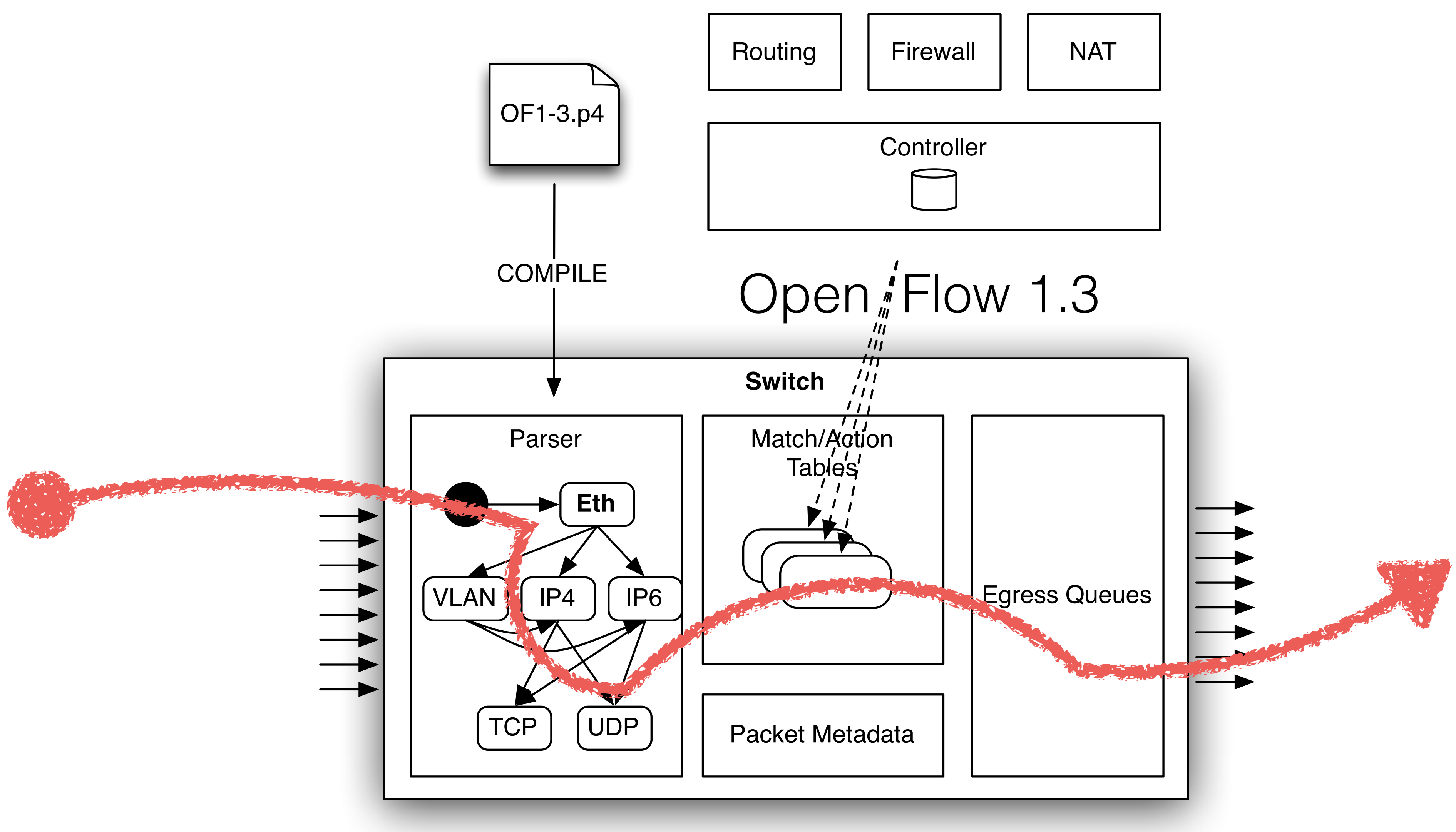
P4 Forwarding Model / Runtime



P4 Forwarding Model / Runtime



P4 Forwarding Model / Runtime



P4 Parsing

```
header vlan {
  fields {
    pcp : 3;
    cfi : 1;
    vid : 12;
    ethertype : 16;
  }
}
```

```
parser start {
  ethernet;
}

parser ethernet {
  switch(ethertype) {
    case 0x8100: vlan;
    case 0x9100: vlan;
    case 0x800: ipv4;
  }
}
```

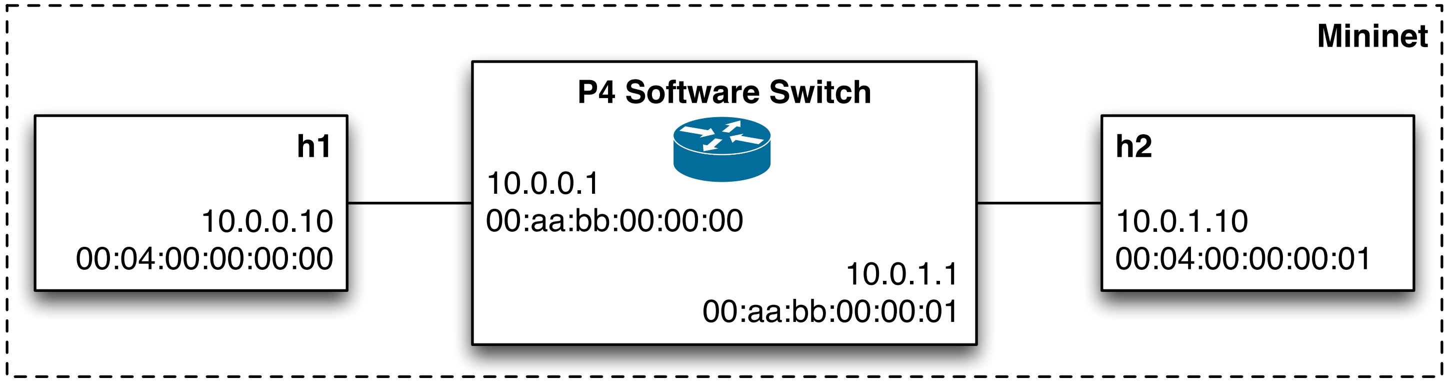
P4 Actions

```
action add_mTag(up1, up2, down1, down2, egr_spec) {
 add_header(mTag);
 copy_field(mTag.ethertype, vlan.ethertype);
 set_field(vlan.ethertype, 0xaaaa);
 set_field(mTag.up1, up1);
 set_field(mTag.up2, up2);
 set_field(mTag.down1, down1);
 set_field(mTag.down2, down2);
}
```

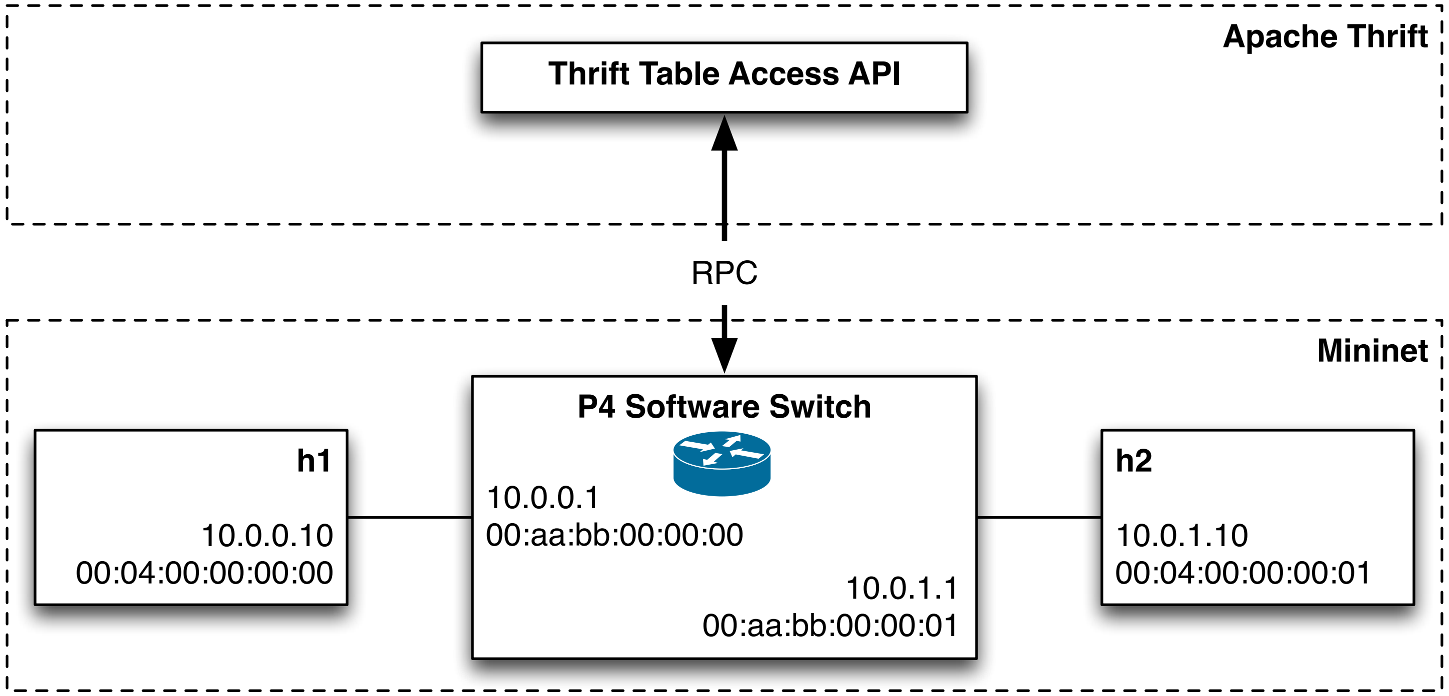
P4 Match/Action

```
table mTag_table {
 reads {
  ethernet.dst_addr : exact; vlan.vid : exact;
 }
 actions {
  add_mTag;
 }
}
```

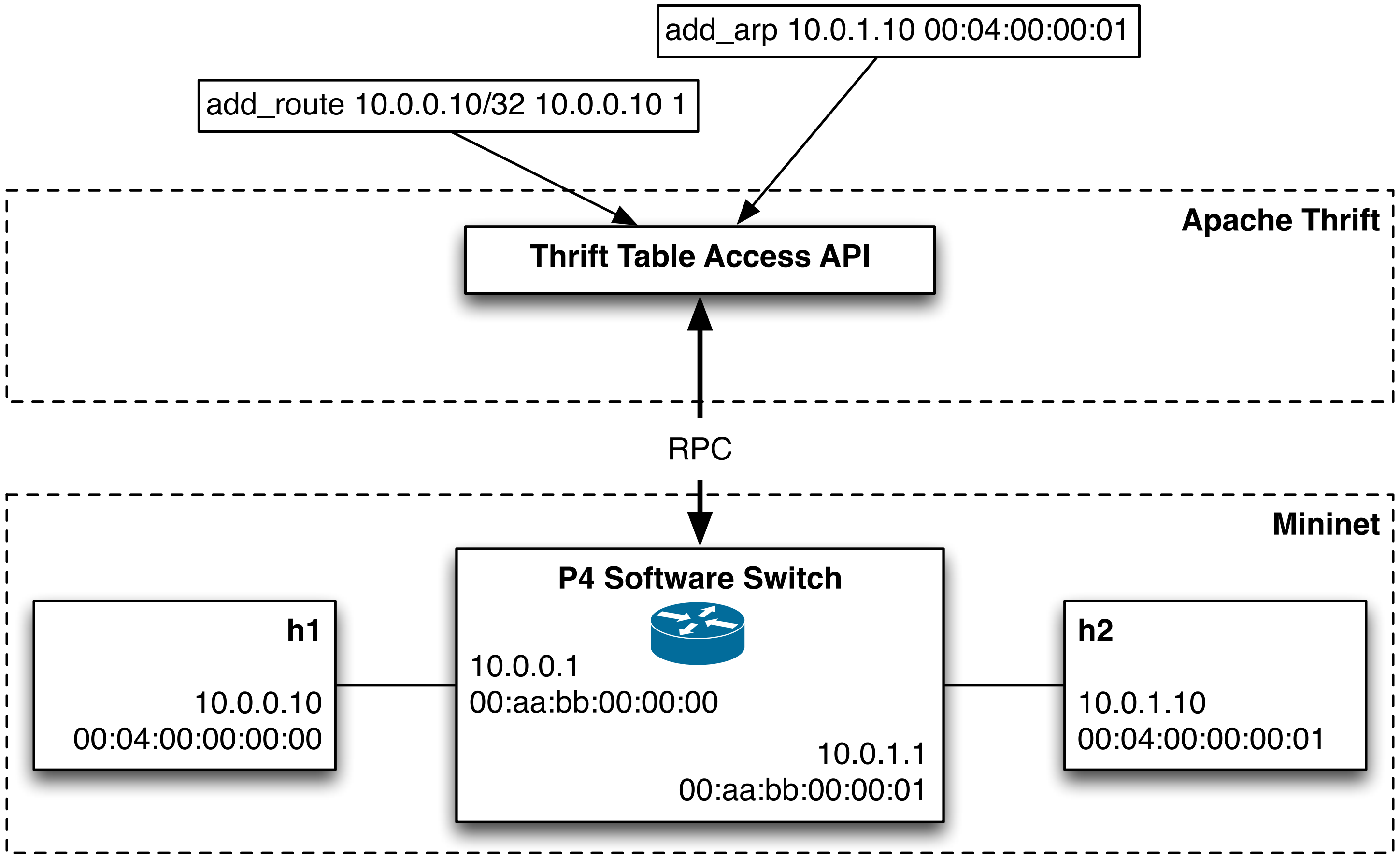
Demo Environment



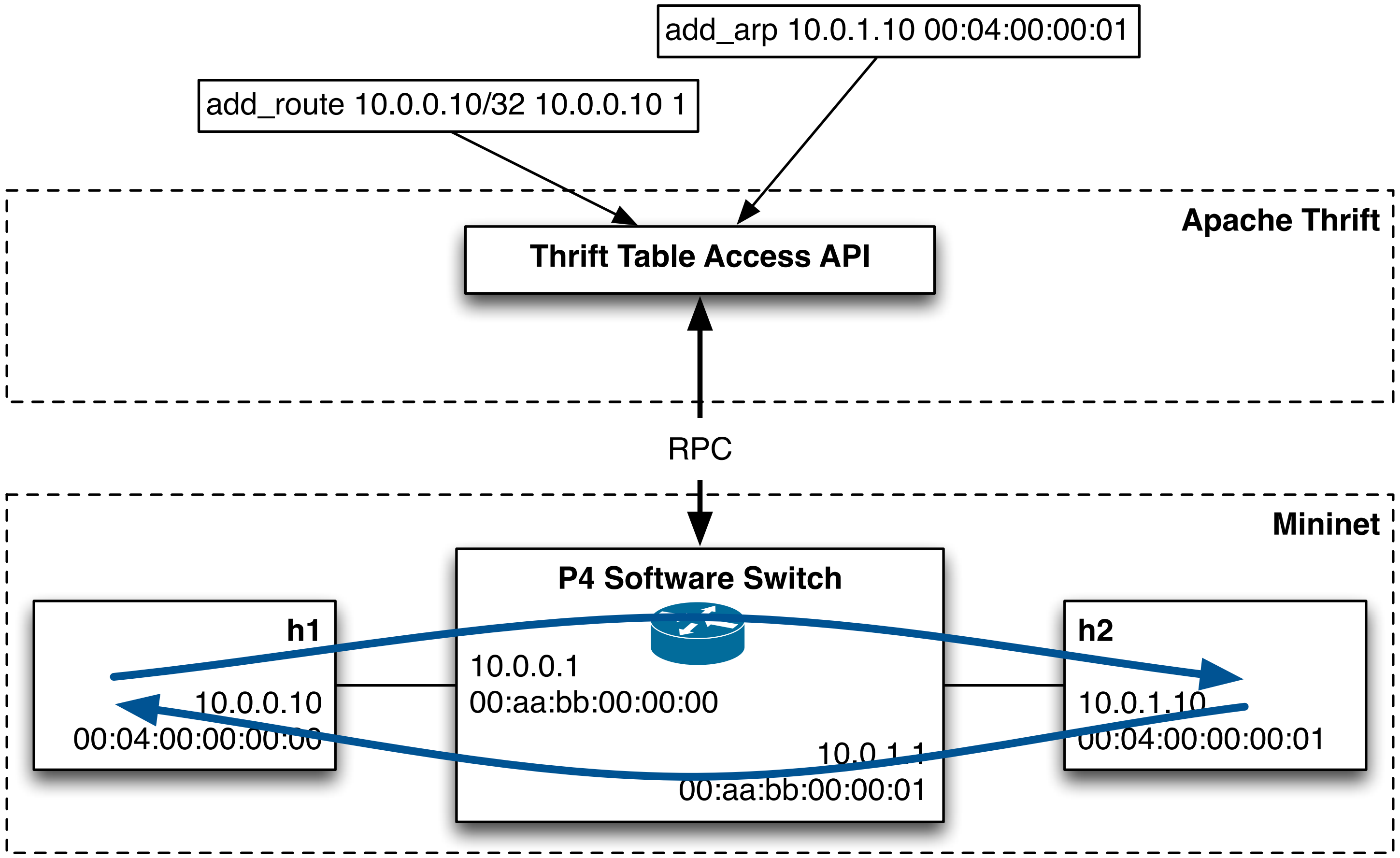
Demo Environment



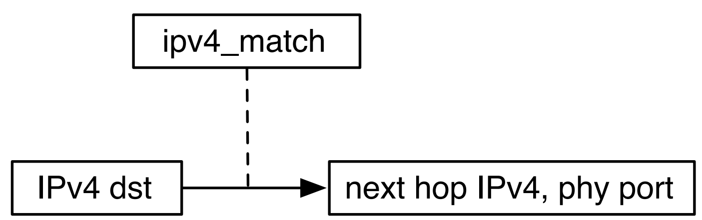
Demo Environment



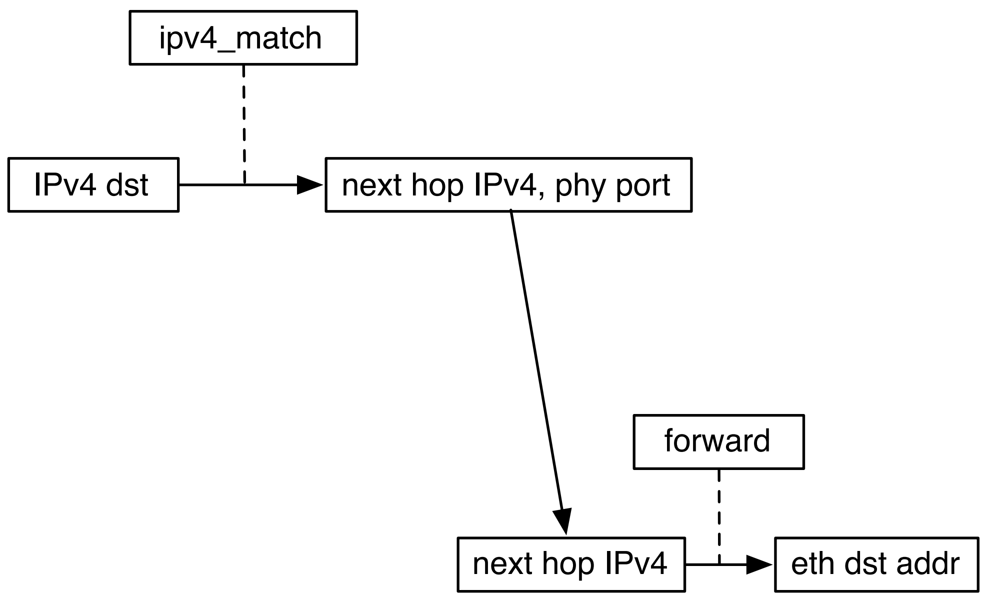
Demo Environment



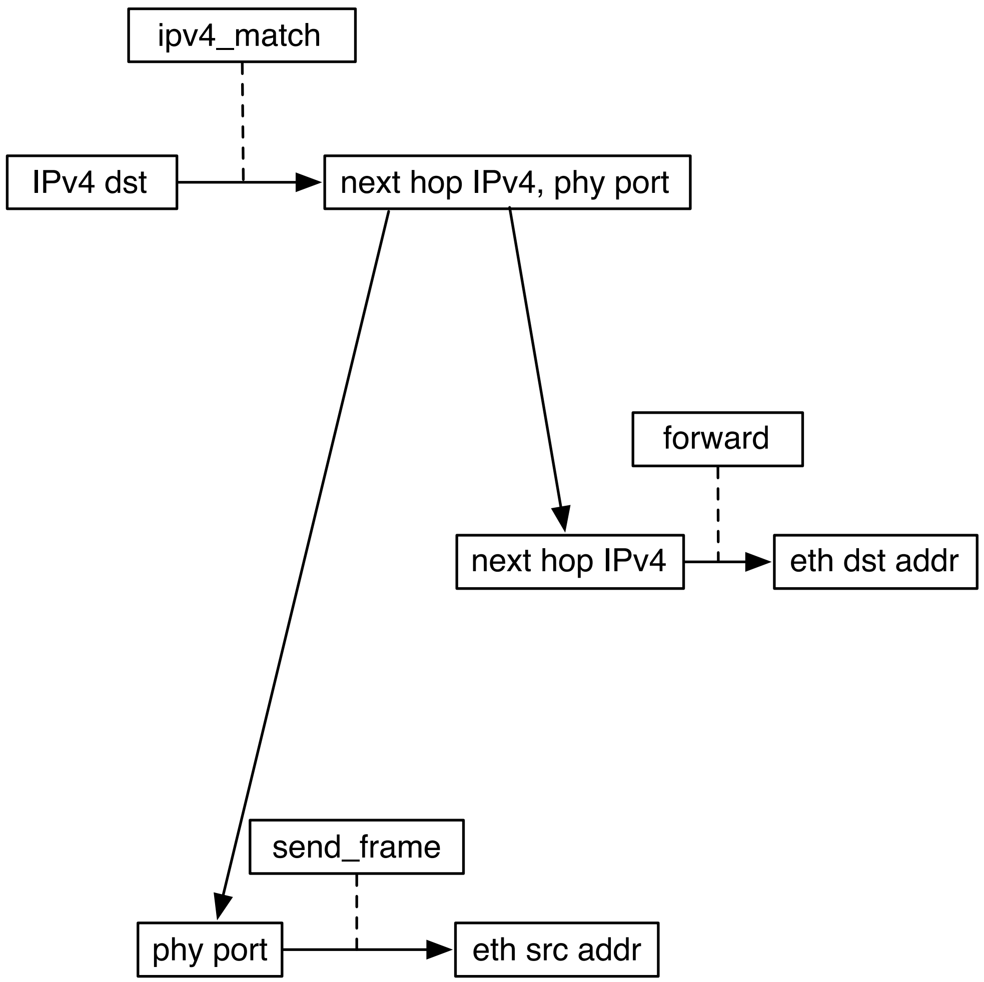
Multiple Tables



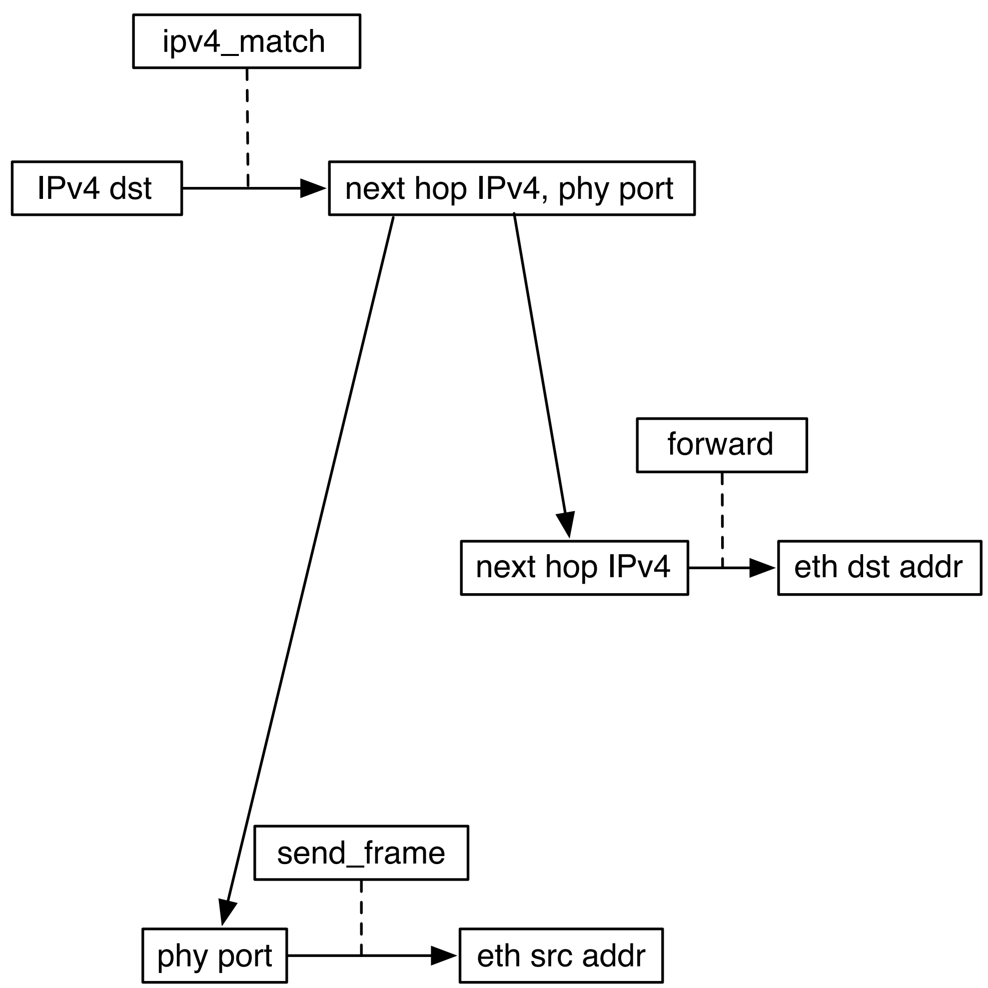
Multiple Tables



Multiple Tables

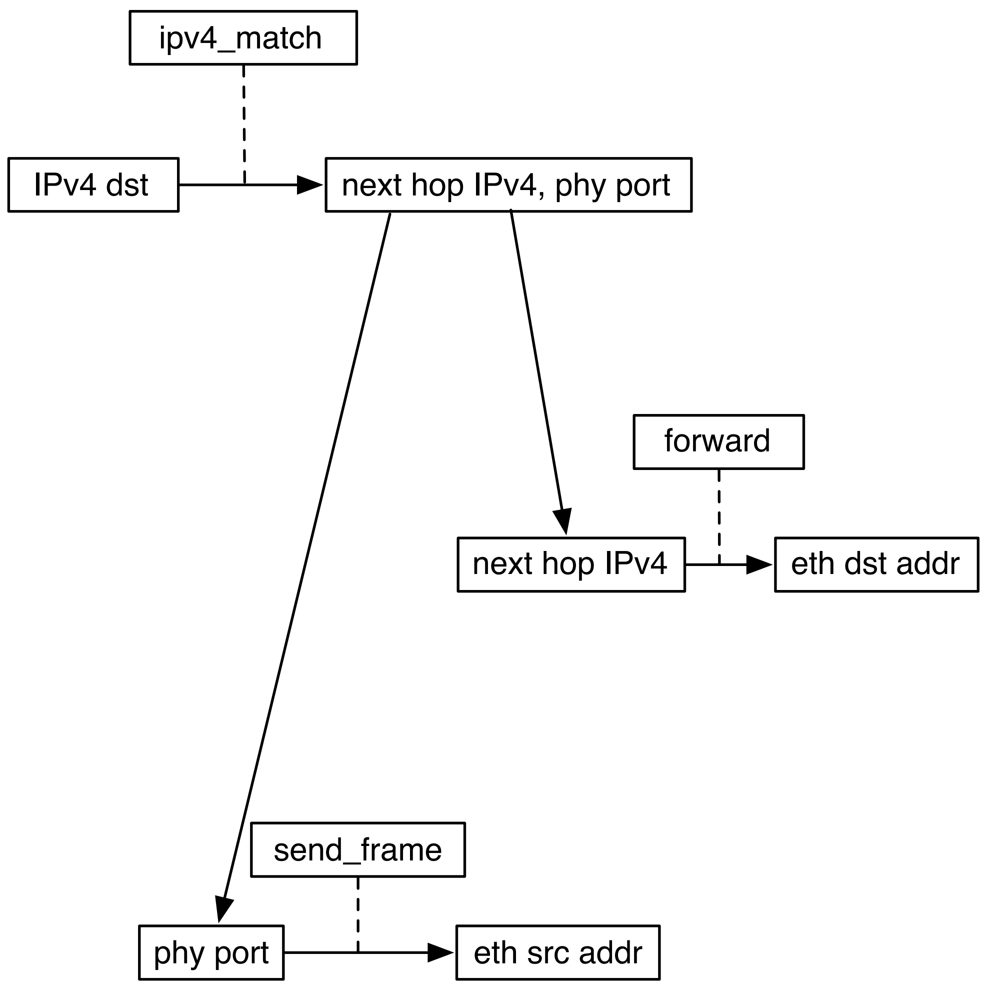


Multiple Tables



10.0.0.10/32	10.0.0.10	1
10.0.1.10/32	10.0.1.10	2

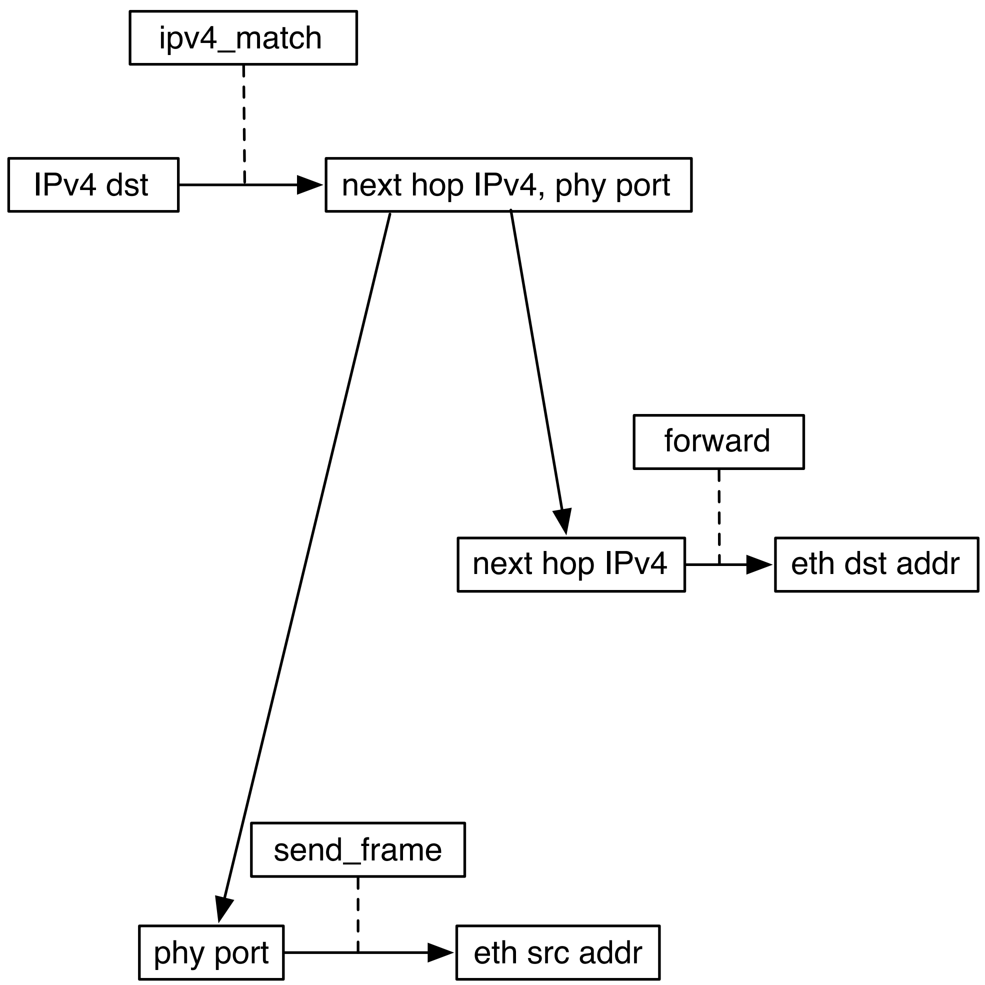
Multiple Tables



10.0.0.10/32	10.0.0.10	1
10.0.1.10/32	10.0.1.10	2

10.0.0.10	00:04:00:00:00:00
10.0.1.10	00:04:00:00:00:01

Multiple Tables



10.0.0.10/32	10.0.0.10	1
10.0.1.10/32	10.0.1.10	2

10.0.0.10	00:04:00:00:00:00
10.0.1.10	00:04:00:00:00:01

1	00:aa:bb:00:00:00
2	00:aa:bb:00:00:01

Parser

```
parser start {
    return parse_ethernet;
}
```

Parser

```
parser start {
    return parse_ethernet;
}
```

```
parser parse_ethernet {
    extract(ethernet);
    return select(latest.etherType) {
        ETHERTYPE_IPV4 : parse_ipv4;
        default: ingress;
    }
}
```

Parser

```
parser start {
    return parse_ethernet;
}
```

```
parser parse_ethernet {
    extract(ethernet);
    return select(latest.etherType) {
        ETHERTYPE_IPV4 : parse_ipv4;
        default: ingress;
    }
}
```

```
parser parse_ipv4 {
    extract(ipv4);
    return ingress;
}
```

Tables

```
table ipv4_match {
    reads {
        ipv4.dstAddr : lpm;
    }
    actions {
        set_nhop;
        _drop;
    }
    size: 1024;
}
```

Tables

```
table ipv4_match {
    reads {
        ipv4.dstAddr : lpm;
    }
    actions {
        set_nhop;
        _drop;
    }
    size: 1024;
}
```

```
table forward {
    reads {
        routing_metadata.nhop_ipv4 : exact;
    }
    actions {
        set_dmac;
        _drop;
    }
    size: 512;
}
```

Tables

```
table ipv4_match {
    reads {
        ipv4.dstAddr : lpm;
    }
    actions {
        set_nhop;
        _drop;
    }
    size: 1024;
}
```

```
table forward {
    reads {
        routing_metadata.nhop_ipv4 : exact;
    }
    actions {
        set_dmac;
        _drop;
    }
    size: 512;
}
```

```
table send_frame {
    reads {
        standard_metadata.egress_port: exact;
    }
    actions {
        rewrite_mac;
        _drop;
    }
    size: 256;
}
```

Tables

```
table ipv4_match {
    reads {
        ipv4.dstAddr : lpm;
    }
    actions {
        set_nhop;
        _drop;
    }
    size: 1024;
}
```

```
table forward {
    reads {
        routing_metadata.nhop_ipv4 : exact;
    }
    actions {
        set_dmac;
        _drop;
    }
    size: 512;
}
```

```
table send_frame {
    reads {
        standard_metadata.egress_port: exact;
    }
    actions {
        rewrite_mac;
        _drop;
    }
    size: 256;
}
```

```
control ingress {
    apply(ipv4_match);
    apply(forward);
}

control egress {
    apply(send_frame);
}
```

Actions

```
action set_nhop(nhop_ipv4, port) {
    modify_field(routing_metadata.nhop_ipv4, nhop_ipv4);
    modify_field(standard_metadata.egress_spec, port);
    add_to_field(ipv4.ttl, -1);
}
```

Actions

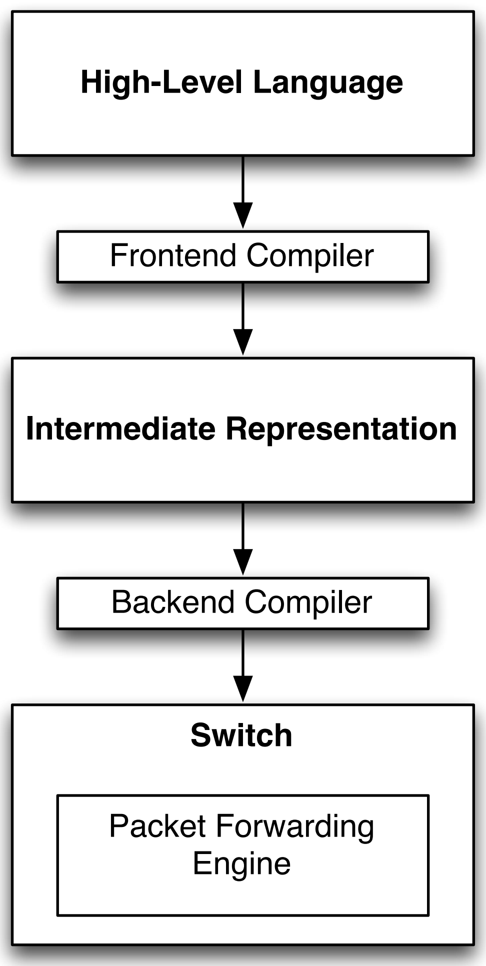
```
action set_nhop(nhop_ipv4, port) {
    modify_field(routing_metadata.nhop_ipv4, nhop_ipv4);
    modify_field(standard_metadata.egress_spec, port);
    add_to_field(ipv4.ttl, -1);
}
```

```
python ../../cli/pd_cli.py
  -p simple_router
  -i p4_pd_rpc.simple_router
  -s $PWD/of-tests/pd_thrift:$PWD/../../submodules/oft-infra
  -m "add_entry ipv4_match 10.0.1.10 32 set_nhop 10.0.1.10 2"
  -c localhost:22222
```

DEMO

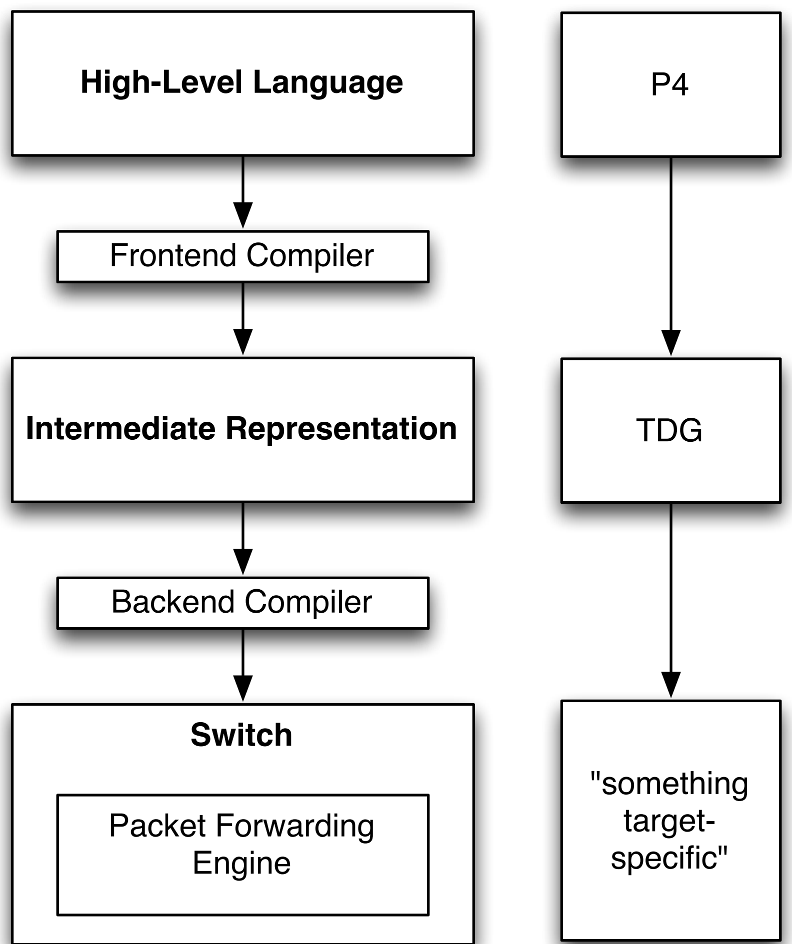
Conclusion / P4 in two slides

Configuration



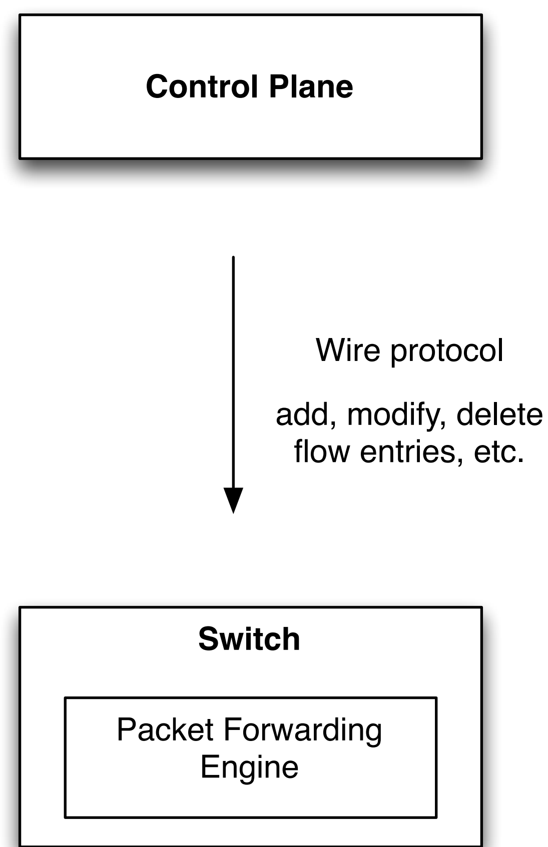
Conclusion / P4 in two slides

Configuration



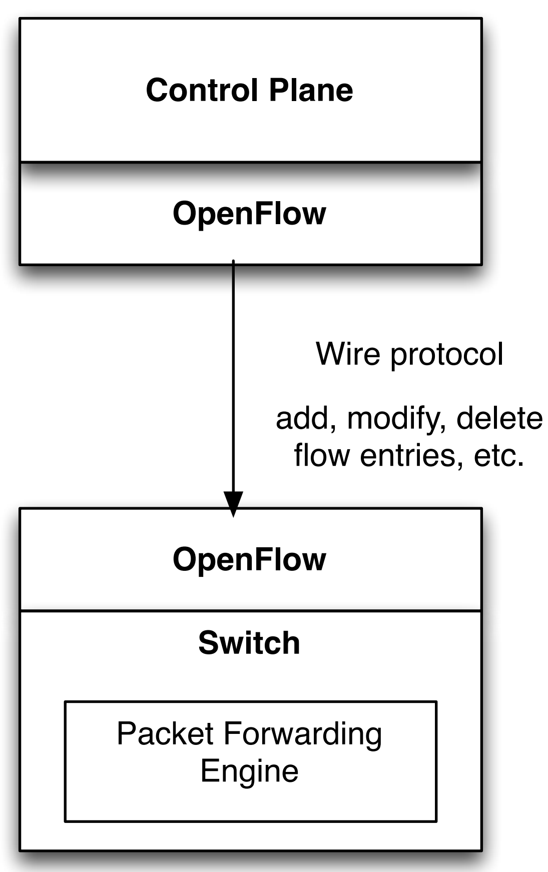
Conclusion / P4 in two slides

Runtime



Conclusion / P4 in two slides

Runtime



DISCUSSION/Q&A

oliver.michel@colorado.edu

BACKUP SLIDES

Control Plane/Data Plane Recap

- Control Plane
  - set up state in routers
  - determines how and where packets are forwarded
- Data Plane
  - actual processing and delivery of packets based on state established by control plane

SDNChip [SIGCOMM 2013]

Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN

Pat Bosshart†, Glen Gibb‡, Hun-Seok Kim†, George Varghese§, Nick McKeown‡,
Martin Izzard†, Fernando Mujica†, Mark Horowitz‡
†Texas Instruments ‡Stanford University §Microsoft Research
pat.bosshart@gmail.com {grg, nickm, horowitz}@stanford.edu
varghese@microsoft.com {hkim, izzard, fmujica}@ti.com

ABSTRACT

In Software Defined Networking (SDN) the control plane is physically separate from the forwarding plane. Control software programs the forwarding plane (e.g., switches and routers) using an open interface, such as OpenFlow. This paper aims to overcomes two limitations in current switching chips and the OpenFlow protocol: i) current hardware switches are quite rigid, allowing "Match-Action" processing on only a fixed set of fields, and ii) the OpenFlow specification only defines a limited repertoire of packet processing actions. We propose the RMT (reconfigurable match tables) model, a new RISC-inspired pipelined architecture for switching chips, and we identify the essential minimal set of action primitives to specify how headers are processed in hardware. RMT allows the forwarding plane to be changed in the field without modifying hardware. As in OpenFlow, the programmer can specify multiple match tables of arbitrary width and depth, subject only to an overall resource limit, with each table configurable for matching on arbitrary fields. However, RMT allows the programmer to modify *all* header fields much more comprehensively than in OpenFlow. Our paper describes the design of a 64 port by 10 Gb/s switch chip implementing the RMT model. Our concrete design demonstrates, contrary to concerns within the community, that flexible OpenFlow hardware switch implementations are feasible at almost no additional cost or power.

Categories and Subject Descriptors

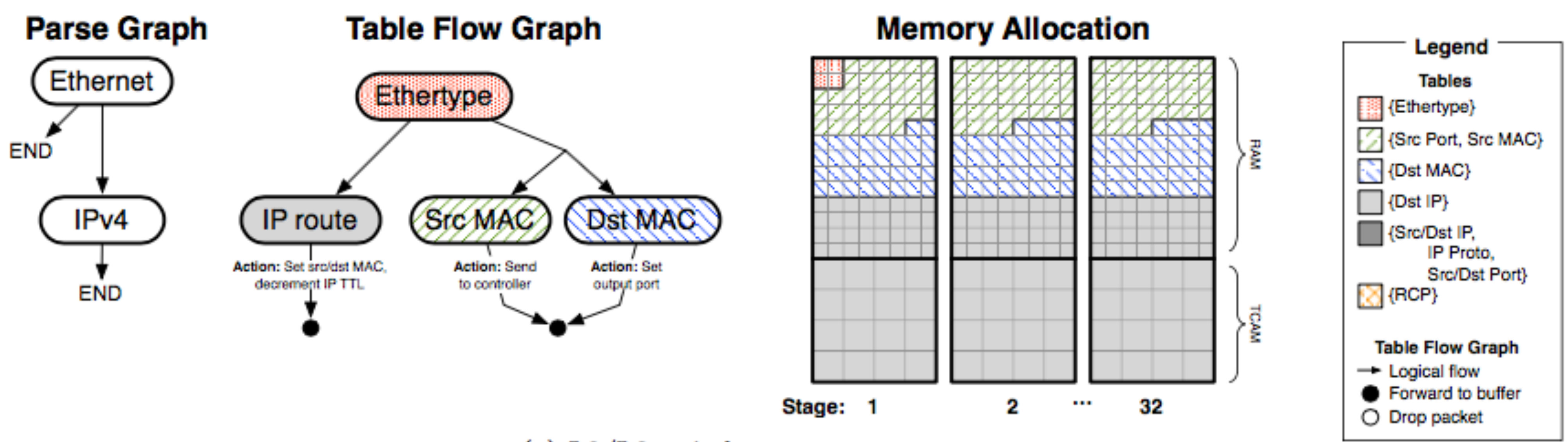
1. INTRODUCTION

To improve is to change; to be perfect is to change often. — Churchill

Good abstractions—such as virtual memory and time-sharing—are paramount in computer systems because they allow systems to deal with change and allow simplicity of programming at the next higher layer. Networking has progressed because of key abstractions: TCP provides the abstraction of connected queues between endpoints, and IP provides a simple datagram abstraction from an endpoint to the network edge. However, routing and forwarding *within* the network remain a confusing conglomerate of routing protocols (e.g., BGP, ICMP, MPLS) and forwarding behaviors (e.g., routers, bridges, firewalls), and the control and forwarding planes remain intertwined inside closed, vertically integrated boxes.

Software-defined networking (SDN) took a key step in abstracting network functions by separating the roles of the control and forwarding planes via an *open* interface between them (e.g., OpenFlow [27]). The control plane is lifted up and out of the switch, placing it in external software. This programmatic control of the forwarding plane allows network owners to add new functionality to their network, while replicating the behavior of existing protocols. OpenFlow has become quite well-known as an interface between the control plane and the forwarding plane based on the approach known as "Match-Action". Roughly, a subset of packet bytes are matched against a table; the matched entry specifies a

SDNChip [SIGCOMM 2013]



(a) L2/L3 switch.

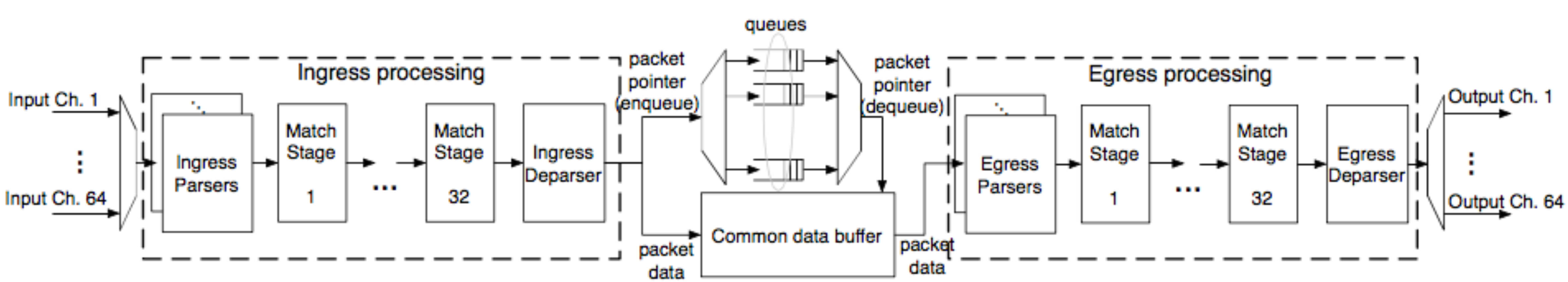


Figure 3: Switch chip architecture.