Programming Network Dataplanes Using P4

Changhoon Kim

P4 Workshop

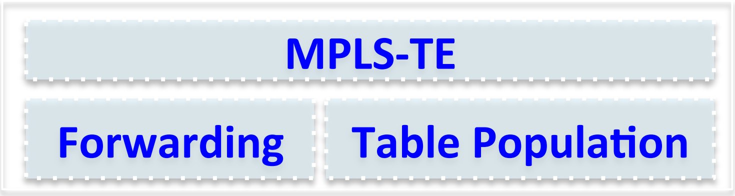
Agenda

- High-level overview of P4
- Current status of P4
  - Language Spec (v1.0.2)
  - Opensource contributions
- Demo
- Exciting use cases

Key players in this new game

- 1. Programmable packet-forwarding solutions
  - S/W switches and GPUs (tens of Gb/s)
  - NPUs and FPGAs (several hundreds of Gb/s)
  - Protocol-independent switching chips (multi Tb/s)
- 2. High-level language
  - P4 (p4.org)
- 3. Compiler and other development tools
  - Common front-end and target-specific back-end compilers
  - Debuggers, profilers, etc.

MPLS (source)

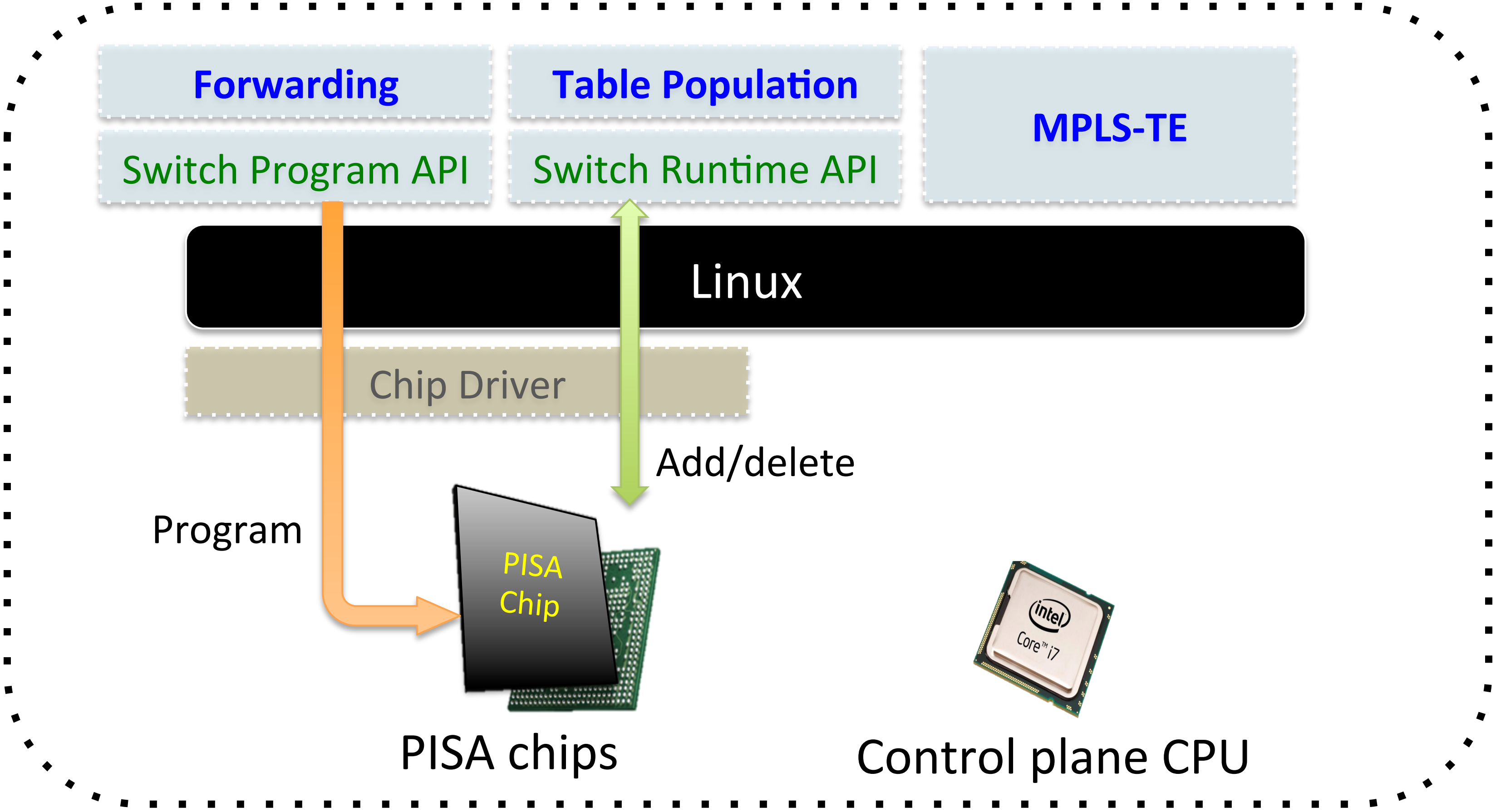


MPLS-TE

Forwarding

Table Population

My (running) switch



Forwarding

Table Population

MPLS-TE

Switch Program API

Switch Runtime API

Linux

Chip Driver

Add/delete

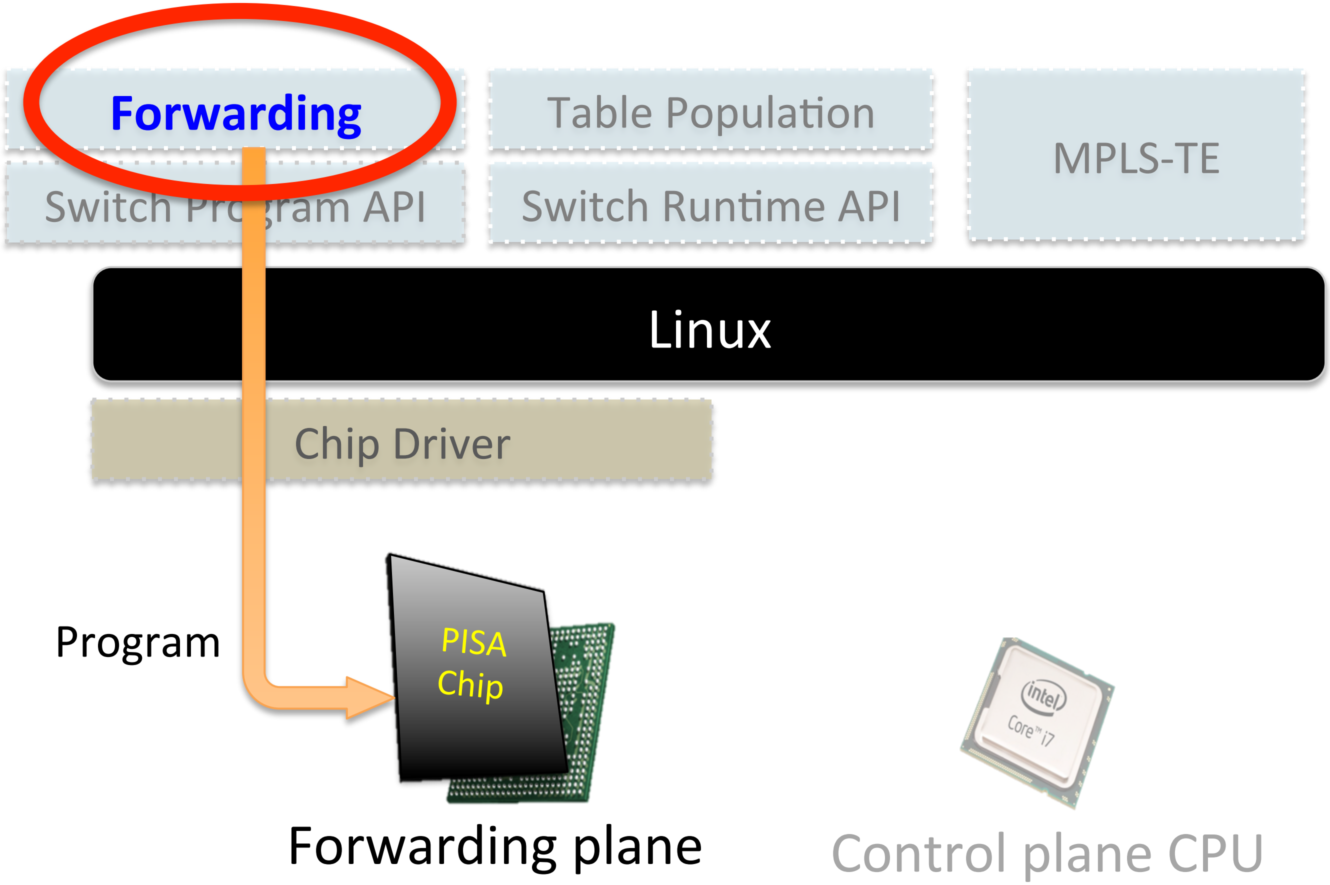
Program

PISA Chip

PISA chips

Control plane CPU

What language to use?



Forwarding

Table Population

MPLS-TE

Switch Program API

Switch Runtime API

Linux

Chip Driver

Program

Forwarding plane

Control plane CPU

P4: Programming Protocol-independent Packet Processors

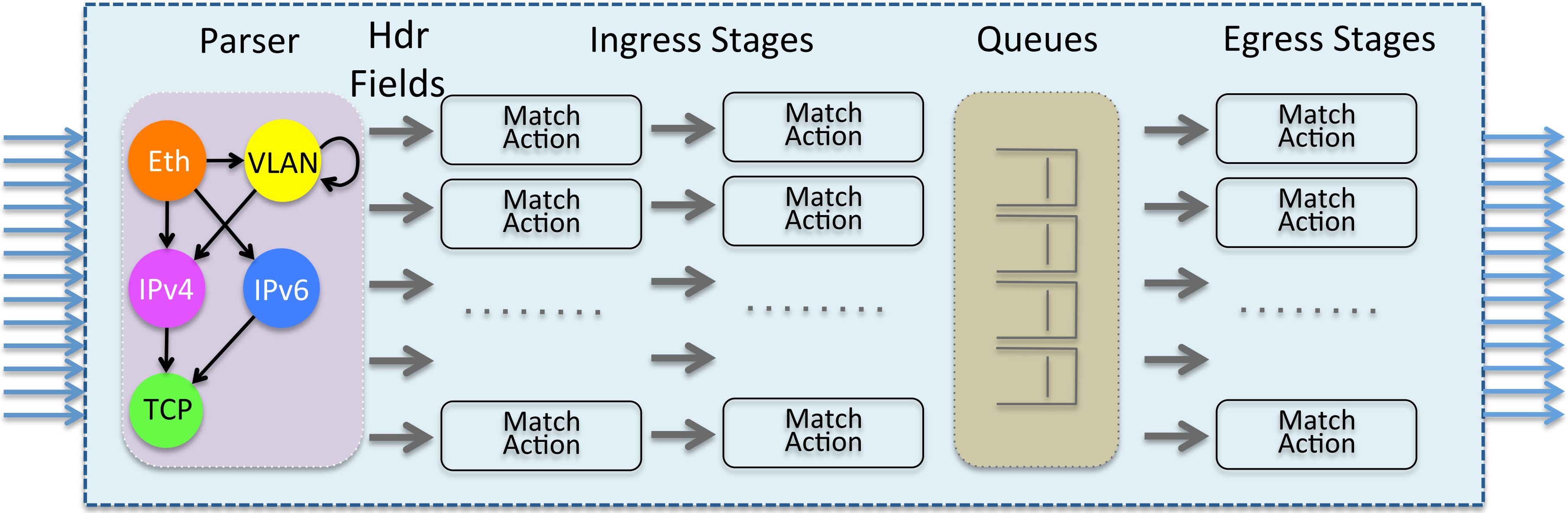
Initial contributions from Stanford, Princeton, Intel, Google, Microsoft, and Barefoot Networks

Goals of P4

- Protocol independence
- Target independence
- Re-configurability in the field

- (And, intuitive abstractions)

Abstract Forwarding Model



P4 Language Components

Parser Program

State-machine;
Field extraction

Match Tables + Actions

Control Flow

Table lookup and update;
Field manipulation;
Control flow

Assembly (“deparser”) Program

Field assembly

No: memory (pointers), loops, recursion, floating point

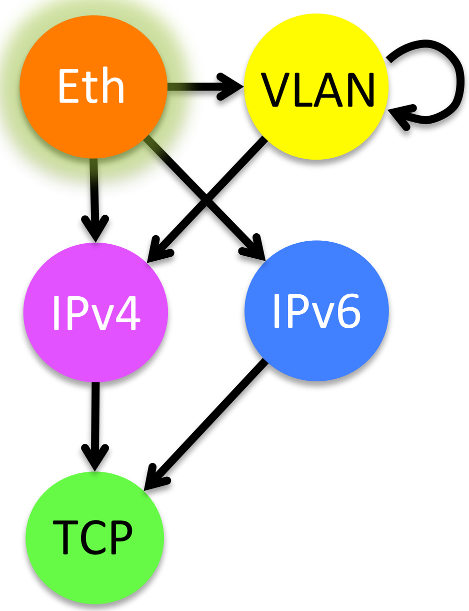
P4 Code Examples

1. Header Fields and Parsing
2. Match + Action Tables
3. Control flow

Header Fields and Parsing

```
header_type ethernet_t {
    fields {
        dstAddr : 48;
        srcAddr : 48;
        etherType : 16;
    }
}
```

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    0x8100 : parse_vlan;
    0x800  : parse_ipv4;
    0x86DD : parse_ipv6;
  }
}
```



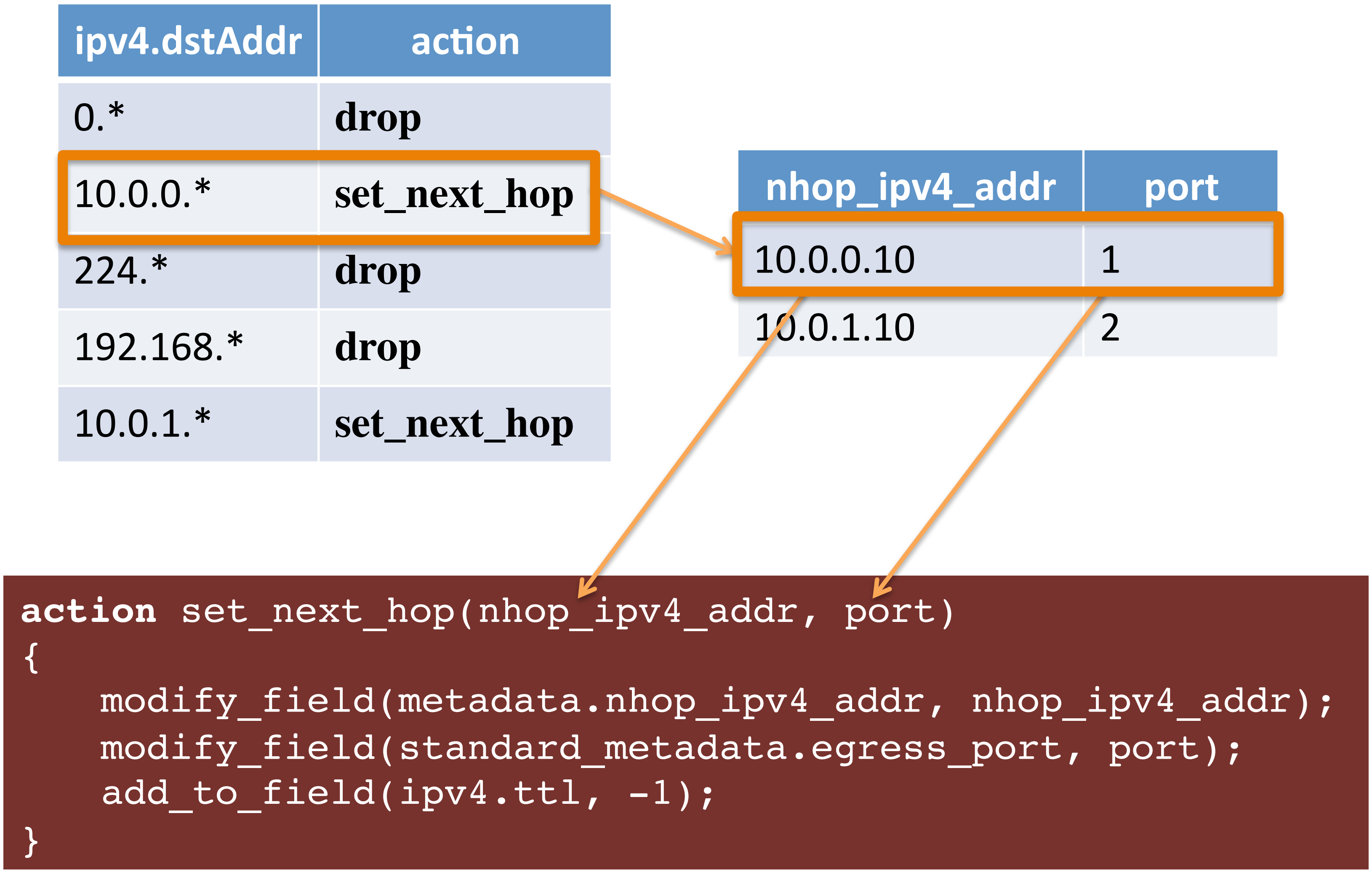
Match

```
table ipv4_lpm
{
   reads {
        ipv4.dstAddr : lpm;
   }
   actions {
        set_next_hop;
        drop;
   }
}
```

Lookup key

ipv4.dstAddr	action
0.*	drop
10.0.0.*	set_next_hop
224.*	drop
192.168.*	drop
10.0.1.*	set_next_hop

Actions

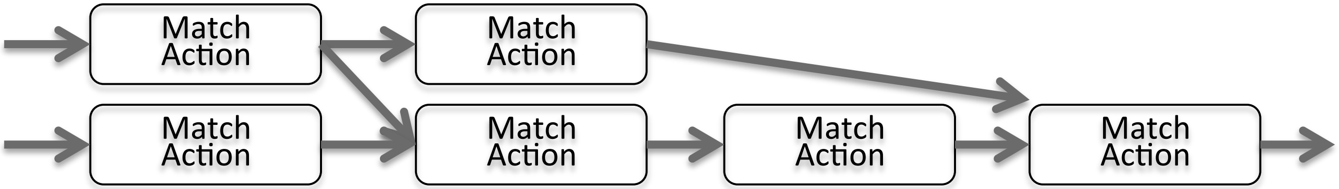


ipv4.dstAddr	action
0.*	drop
10.0.0.*	set_next_hop
224.*	drop
192.168.*	drop
10.0.1.*	set_next_hop

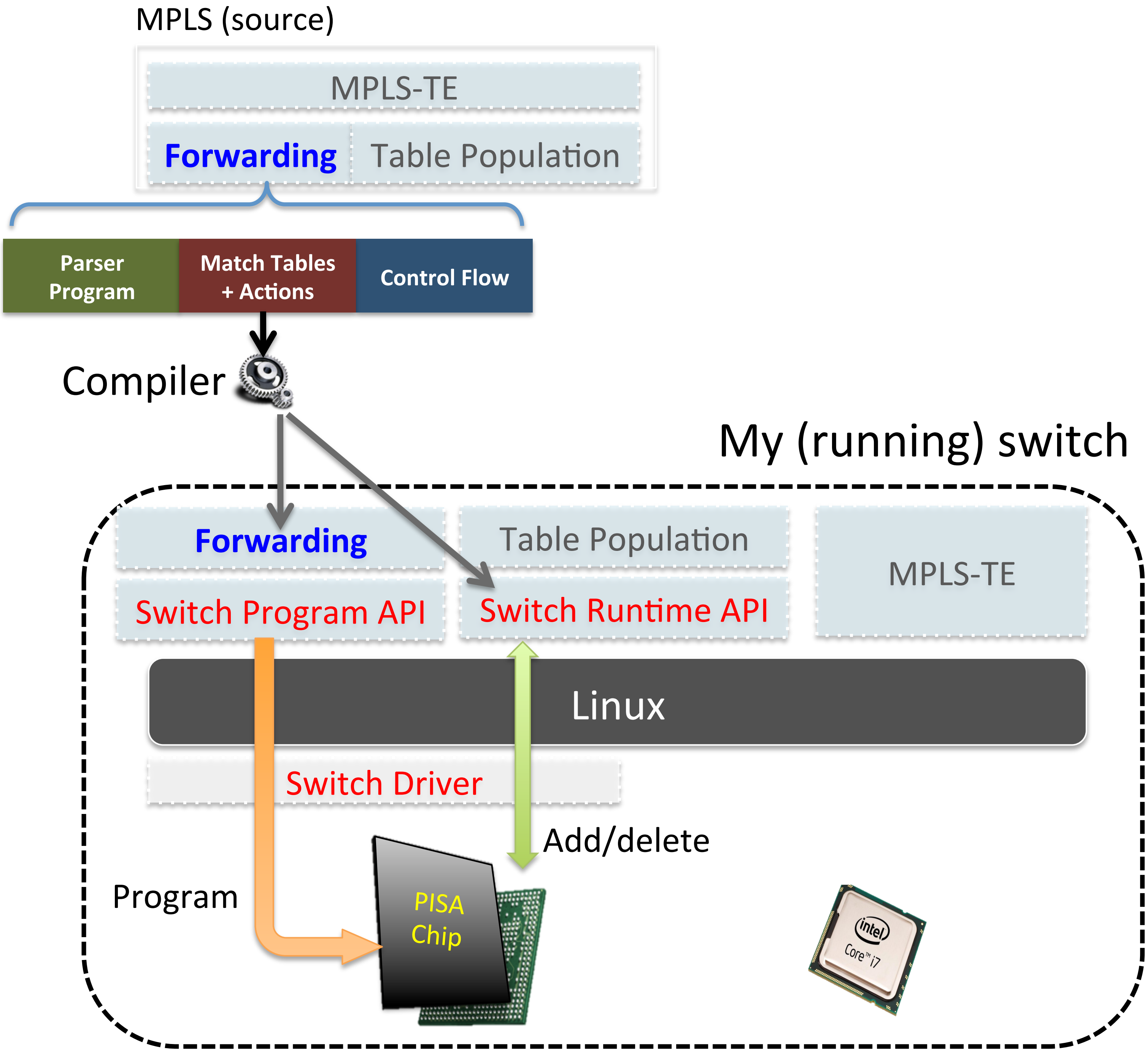
nhop_ipv4_addr	port
10.0.0.10	1
10.0.1.10	2

```
action set_next_hop(nhop_ipv4_addr, port)
{
    modify_field(metadata.nhop_ipv4_addr, nhop_ipv4_addr);
    modify_field(standard_metadata.egress_port, port);
    add_to_field(ipv4.ttl, -1);
}
```

Control Flow



```
control ingress
{
    apply(port);
    if (valid(vlan_tag[0])) {
        apply(port_vlan);
    }
    apply (bridge_domain);
    if (valid(mpls_bos)) {
        apply(mpls_label);
    }
    retrieve_tunnel_vni();
    if (valid(vxlan) or valid(genv) or valid(nvgre))
    {
        apply(dest_vtep);
        apply(src_vtep);
    }
}
```



MPLS (source)

MPLS-TE

Forwarding

Table Population

Parser Program

Match Tables + Actions

Control Flow

Compiler

My (running) switch

Forwarding

Table Population

MPLS-TE

Switch Program API

Switch Runtime API

Linux

Switch Driver

Add/delete

Program

PISA Chip

Switch.p4: A datacenter switch in P4

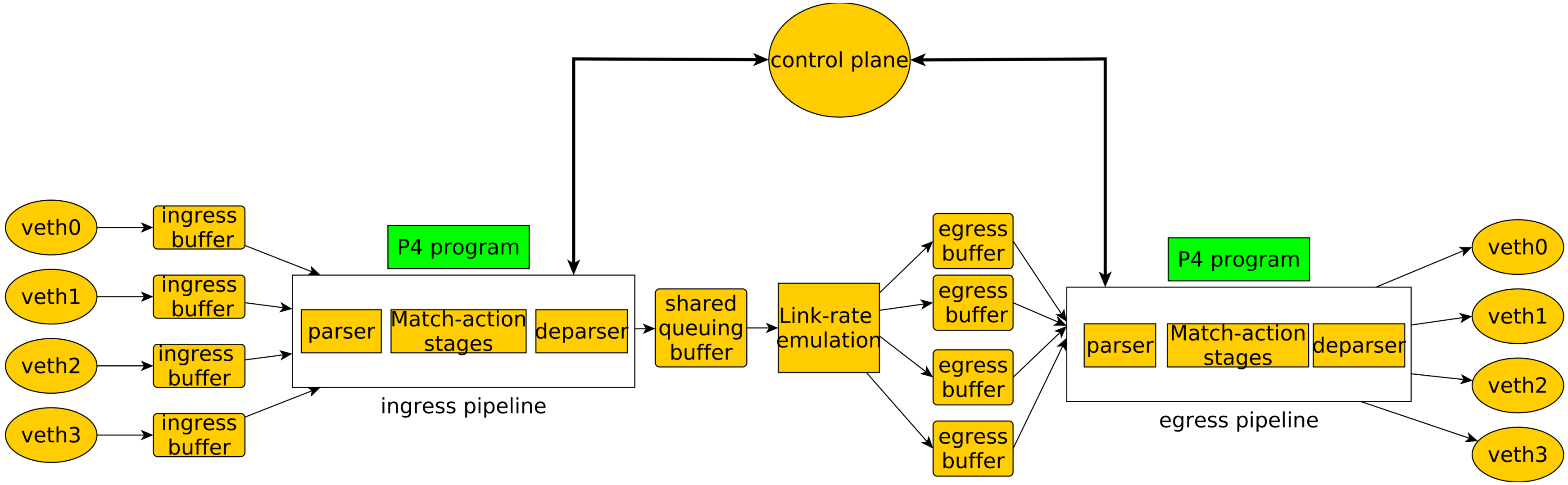
- Can we express the forwarding plane of a typical DC switch in P4?
- Feature set
  - Basic L2 switching: MAC learning, VLAN, flooding, and STP
  - Basic L3 routing: IPv4 and IPv4 routing with VRF
  - LAG
  - ECMP
  - Tunneling: VXLAN and NVGRE (L2/L3 Gateway), Geneve, and GRE
  - Basic ACL: MAC and IP ACLs
  - Unicast RPF
  - MPLS: LER, LSR, IPVPN, VPLS, and L2VPN
- More features coming soon

Opensource P4 Dev Environment

- Sample P4 programs
- Compiler
- S/W P4 switch
- Test framework

S/W P4 Switch

- Stand-alone test framework
- Network-level test framework
  - Pluggable into Mininet through virtual interfaces



Original P4 Paper → Spec v1.0.2

- Primitive actions
  - clone() variants
  - drop()
  - generate_digest()
  - modify_field_with_hash()
- Action profile
- Field list
- Header stacks
- Counters, meters, and registers

New Language Constructs Enabled More Features

- ECMP
- MAC learning
- ACLs
- Mirroring
- MPLS
- Statistics
- Various hash-based algorithms

Avenues Where More Contributions Are Needed

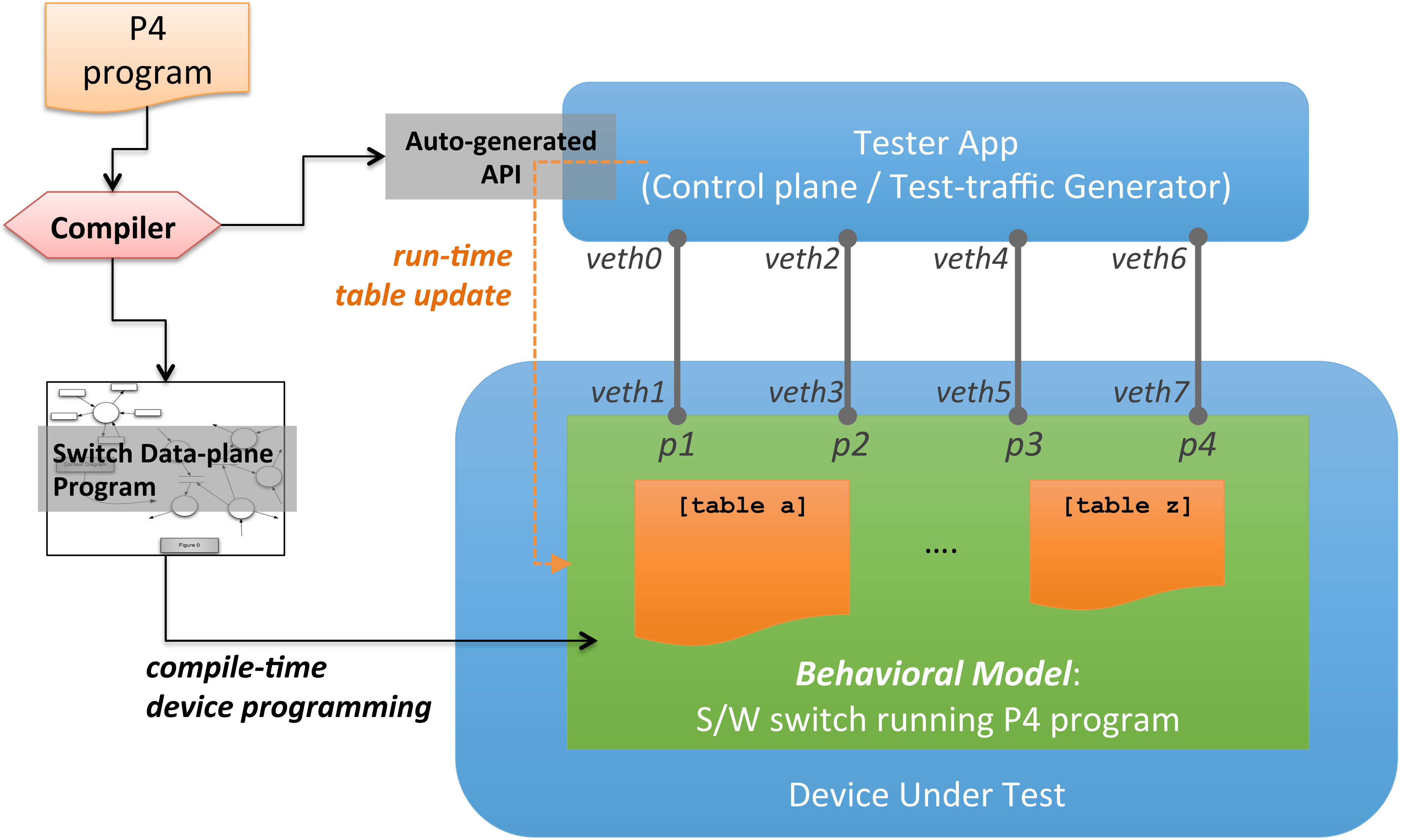
- Short term
  - Constructs for TLV-style header parsing
  - Register manipulation

- Long term
  - Modeling buffers and schedulers
    - Buffer management schemes
      - Carving up buffers into queues and application pools
    - Packet scheduling schemes
      - PQ, WFQ, HFQ, etc.

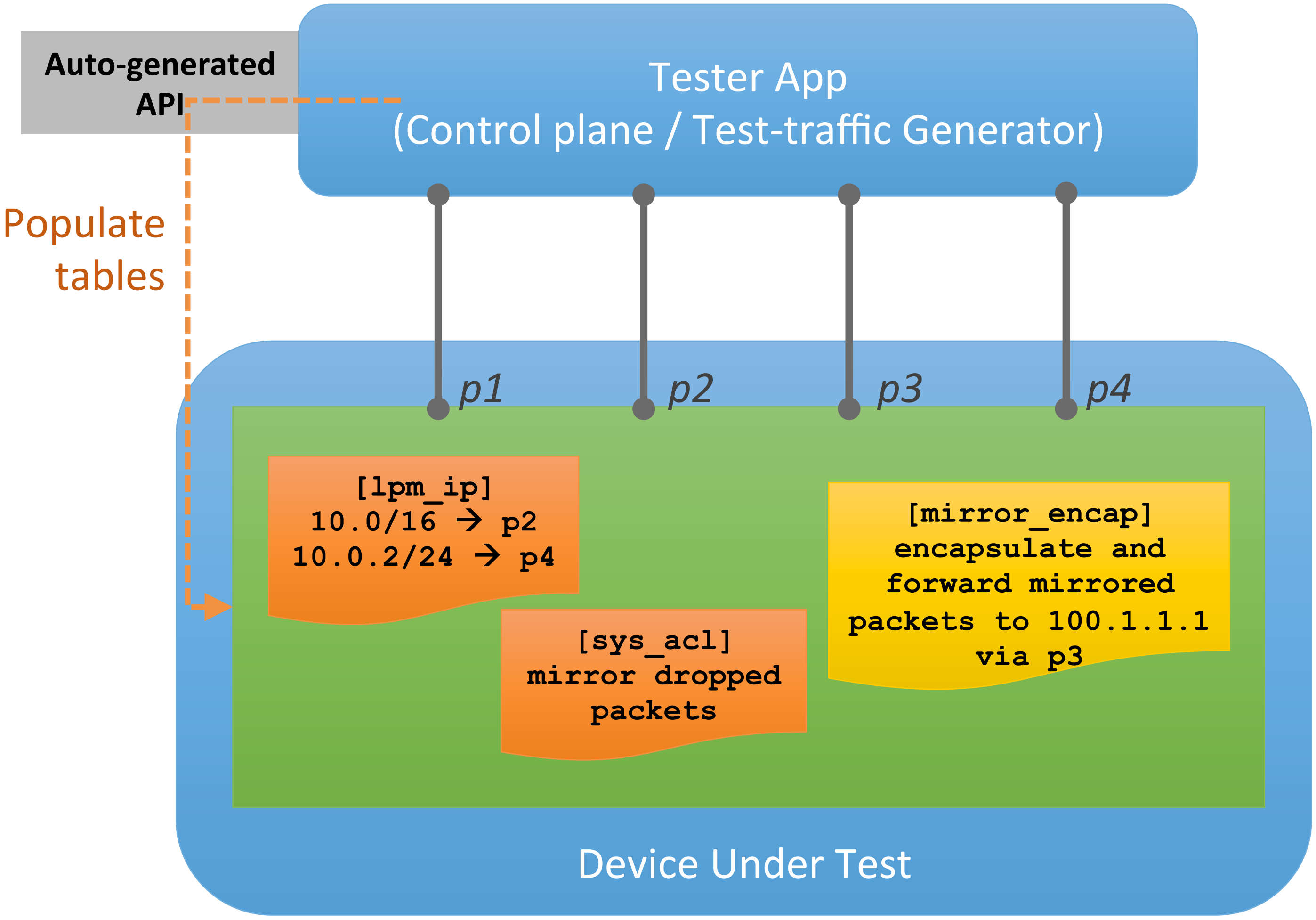
DEMO!

Reproducible via opensource dev tools and reference P4 programs

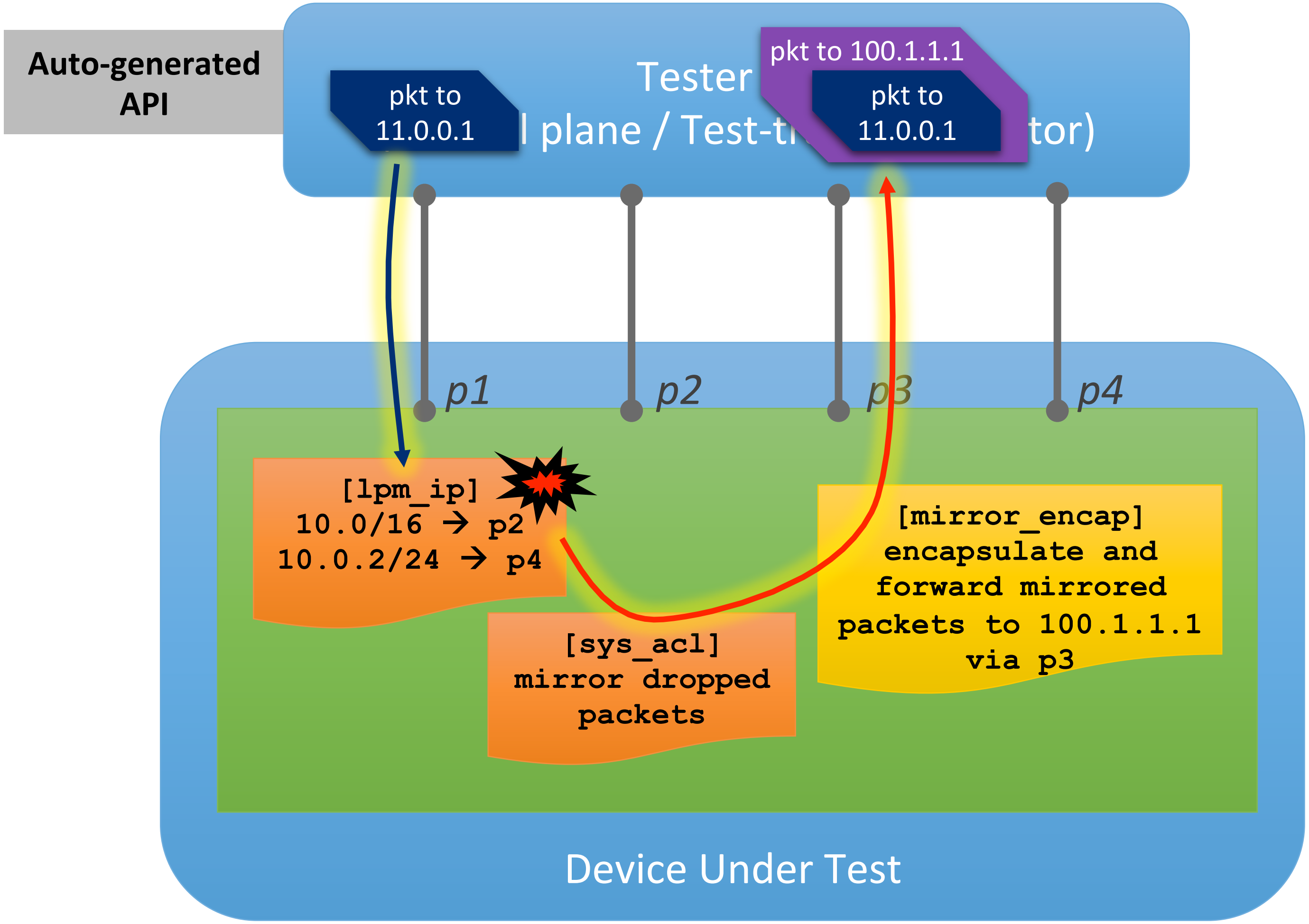
Demo Setup


P4 program
Compiler
Auto-generated API
run-time table update
Switch Data-plane Program
Context Diagram
Figure 0
compile-time device programming
Tester App
(Control plane / Test-traffic Generator)
veth0
veth2
veth4
veth6
veth1
veth3
veth5
veth7
p1
p2
p3
p4
[table a]
....
[table z]
Behavioral Model:
S/W switch running P4 program
Device Under Test

Scenario 1: Mirroring Dropped Packets



Scenario 1: Mirroring Dropped Packets



Auto-generated API

Tester

l plane / Test-t tor)

pkt to 11.0.0.1

pkt to 100.1.1.1

pkt to 11.0.0.1

p1

p2

p3

p4

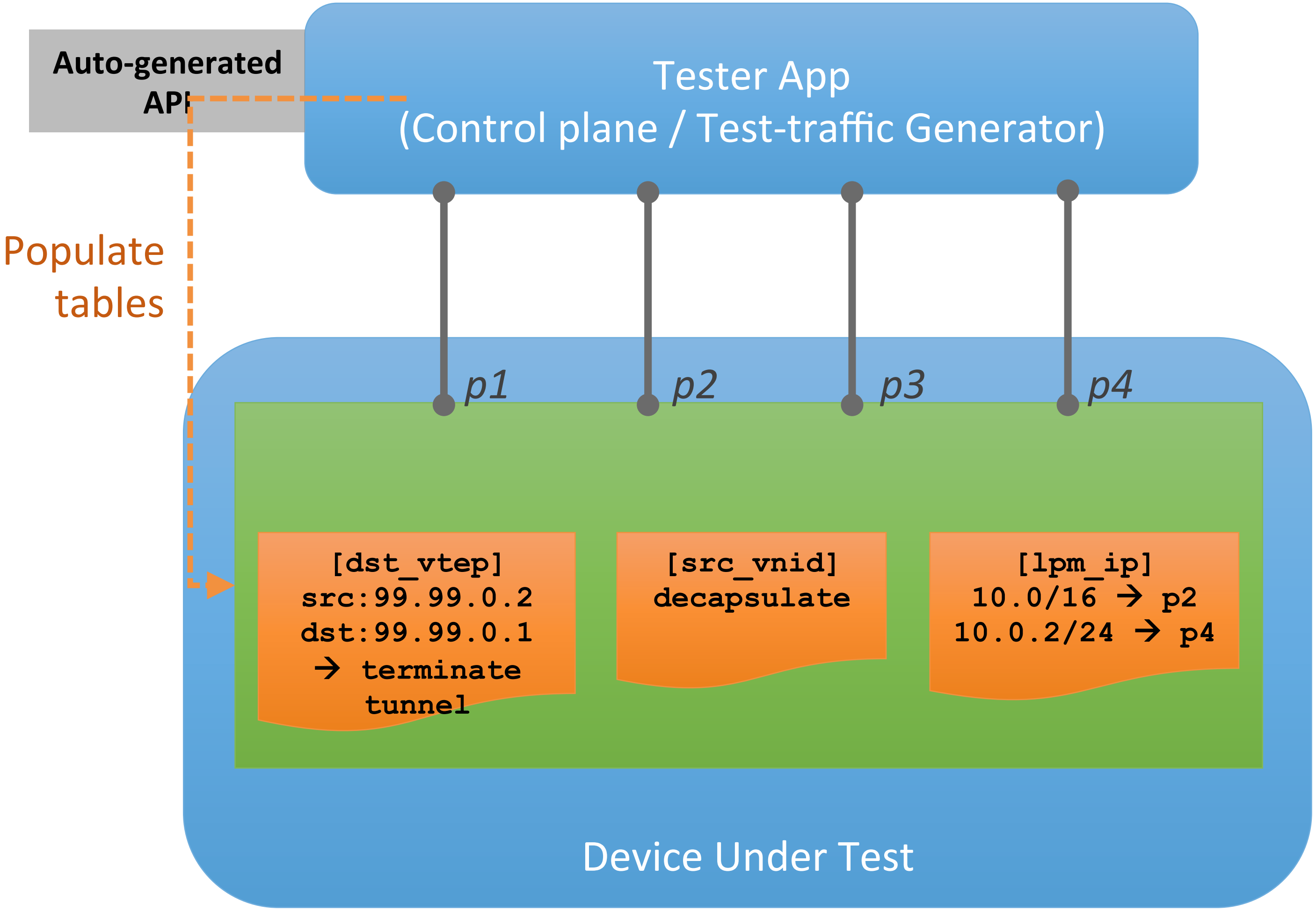
[lpm_ip]
10.0/16 → p2
10.0.2/24 → p4

[sys_acl]
mirror dropped packets

[mirror_encap]
encapsulate and forward mirrored packets to 100.1.1.1 via p3

Device Under Test

Scenario 2: VXLAN Routing

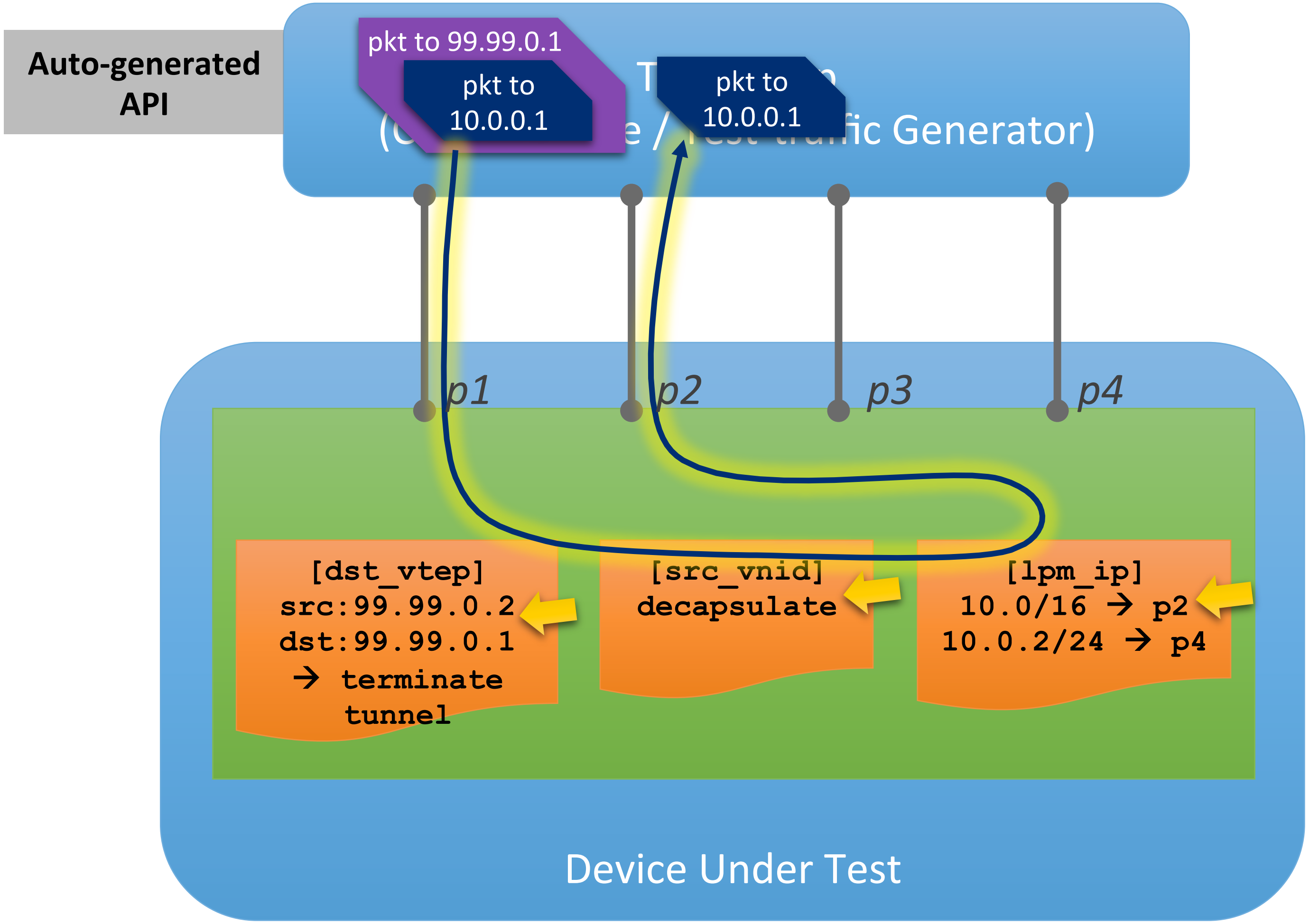


```
[dst_vtep]
src:99.99.0.2
dst:99.99.0.1
→ terminate
tunnel
```

```
[src_vnid]
decapsulate
```

```
[lpm_ip]
10.0/16 → p2
10.0.2/24 → p4
```

Scenario 2: VXLAN Routing



Exciting Applications

- In-band Network Telemetry (INT)
  - Highly user for debugging and analysis
- Tunnel splicing
  - P2V gateways
- L4 load-balancing
- Utilization-aware fabric load-balancing

INT Specification

- INT v1.5 is available to P4.org members
  - Includes reference P4 code
- Participants
  - VMware, Barefoot, and a few other companies
- We would appreciate feedback from the P4 community

Summary

- Common industry-wide forwarding language will be immensely helpful
- P4 looks very promising
- How can you get involved?
  - Join P4.org
  - Assign engineers to get familiar with P4
  - Start playing tools, language, and sample P4 code
  - Contribute back to P4.org

Full-day tutorial at SIGCOMM in Aug at London!

SIGCOMM 2015