Introduction to P4

Programming Protocol-Independent Packets Processors

Ronald van der Pol

SURFnet

(Ronald.vanderPol@rvdp.org)

Programmable Dataplanes

- Two emerging important concepts:
  - Disaggregation
    - De-coupling of switch/router hardware and firmware
    - Already happening with white label/bare metal 1U data centre switches
  - Programmable Dataplanes
    - BUT programmable by end-user/operator!
    - Well-known example: OpenFlow
    - Next step: P4

Disaggregation

- White label/bare metal switch vendors:
  - Quanta, Accton/Edge-Core, DNI/Agema, Dell
  - Boot firmware via ONIE boot loader
- Software/Firmware/(N)OS:
  - Pica8 PicOS (L2/L3 & OpenFlow) commercial
  - Cumulus (L2/L3) commercial
  - Open Network Linux (L2/L3 & OpenFlow)
    - Work in Progress
    - Open Source - part of Open Compute Project

Programmable Dataplanes

- End-user/operator has full access to forwarding pipeline
- OpenFlow is a well known example
- OpenFlow is currently supported on hardware switches with fixed pipeline ASICs (inflexible/ limited functionality)
- P4 is the next step (create parser, lookup tables and pipeline and load it on the switch)

P4 Targets

- Switch with programmable silicon (Barefoot?)
- NIC with programmable silicon (Netronome?)
- FPGA switch (Corsa prototype)
- Software switch (P4 comes with ready to use reference implementation)
- etc

Why is this important?

- Minimize dependency on vendor roadmaps
- Enhance an existing protocol (e.g. metering)
- Define and implement a new protocol (fast prototyping on real hardware)

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[**]
[†]Barefoot Networks [*]Intel [‡]Stanford University [**]Princeton University [¶]Google [§]Microsoft Research

ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are deployed. (2) Protocol independence: Switches should not be tied to any specific network protocols. (3) Target independence: Programmers should be able to describe packet-processing functionality independently of the specifics of the underlying hardware. As an example, we describe how to use P4 to configure a switch to add a new hierarchical label.

1. INTRODUCTION

Software-Defined Networking (SDN) gives operators programmatic control over their networks. In SDN, the control plane is physically separate from the forwarding plane, and one control plane controls multiple forwarding devices. While forwarding devices could be programmed in many ways, having a common, open, vendor-agnostic interface (like OpenFlow) enables a control plane to control forwarding devices from different hardware and software vendors.

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

Table 1: Fields recognized by the OpenFlow standard

The OpenFlow interface started simple, with the abstraction of a single table of rules that could match packets on a dozen header fields (e.g., MAC addresses, IP addresses, protocol, TCP/UDP port numbers, etc.). Over the past five years, the specification has grown increasingly more complicated (see Table 1), with many more header fields and multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open interface (i.e., a new "OpenFlow 2.0" API). Such a general, extensible approach would be simpler, more elegant, and more future-proof than today's OpenFlow 1.x standard.

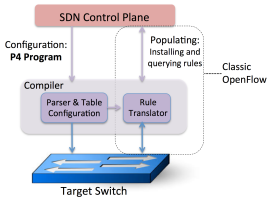


Figure 1: P4 is a language to configure switches.

Recent chip designs demonstrate that such flexibility can be achieved in custom ASICs at terabit speeds [1, 2, 3]. Programming this new generation of switch chips is far from easy. Each chip has its own low-level interface, akin to microcode programming. In this paper, we sketch the design of a higher-level language for Programming Protocol-independent Packet Processors (P4). Figure 1 shows the relationship between P4—used to configure a switch, telling it how packets are to be processed—and existing APIs (such as OpenFlow) that are designed to populate the forwarding tables in fixed function switches. P4 raises the level of abstraction for programming the network, and can serve as a

P4 Language Consortium

- Two board members:
  - Nick McKeown (Stanford University)
  - Jennifer Rexford (Princeton University)
- ~ 32 industry members
- ~ 6 university members

INDUSTRY MEMBERS

MoSys

NETRONOME

NoviFlow
SDN made smarter

PLUMgrid

SK telecom

Tencent 腾讯

vmware

XILINX

UNIVERSITY MEMBERS

CORNELL UNIVERSITY FOUNDED A.D. 1865

POLITECNICO MILANO 1863

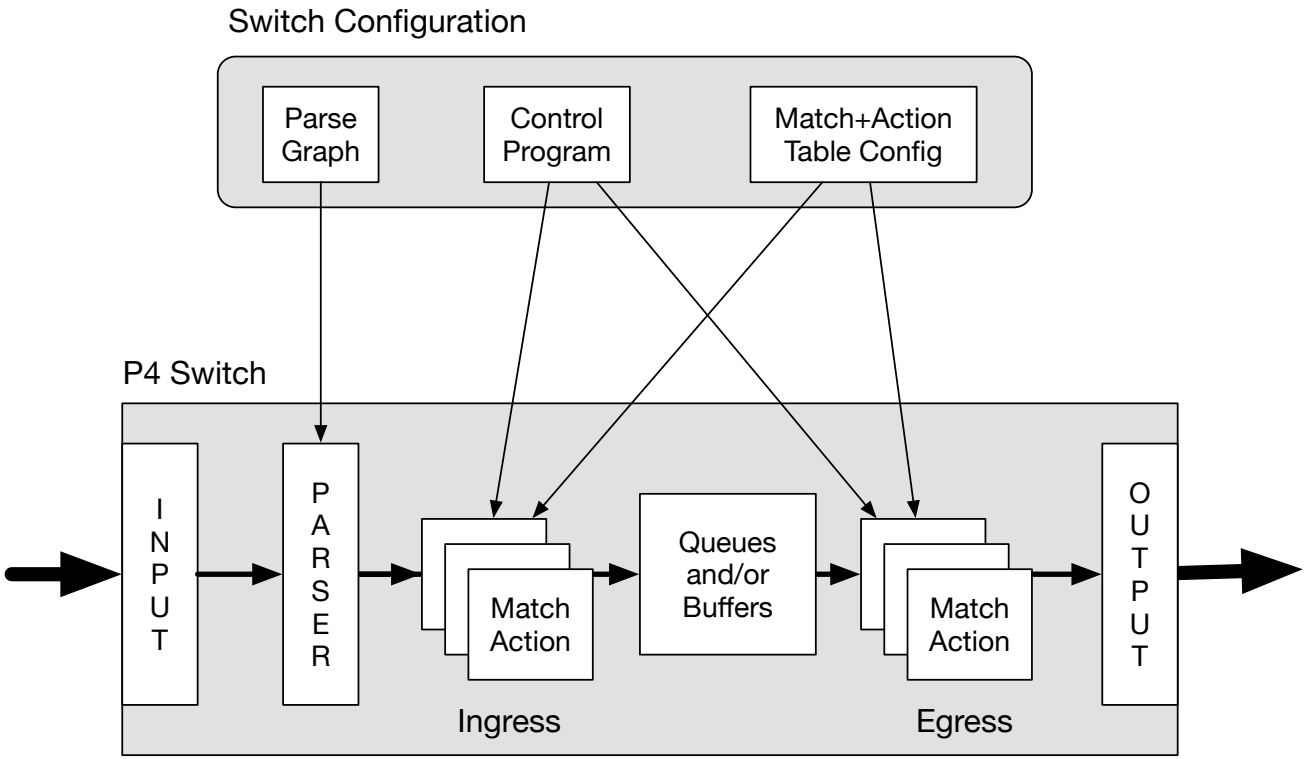
PRINCETON UNIVERSITY

Stanford University

UNIVERSITÉ DU LUXEMBOURG

Università della Svizzera italiana

P4 Switch



Source: The P4 Language Specification Version 1.0.2

P4 Eth & IPv4 Header Definitions

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

```
header_type ipv4_t {
   fields {
      version : 4;
      ihl : 4;
      diffserv : 8;
      totalLen : 16;
      identification : 16;
      flags : 3;
      fragOffset : 13;
      ttl : 8;
      protocol : 8;
      hdrChecksum : 16;
      srcAddr : 32;
      dstAddr: 32;
   }
}
```

P4 VLAN Header Definition

```
header_type vlan_t {
    fields {
        pcp : 3;
        cfi : 1;
        vid : 12;
        ethertype : 16;
    }
}
```

P4 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;
}
```

P4 IPv4 FIB Table Example

```
table ipv4_fib_lpm {
    reads {
        ingress_metadata.vrf : exact;
        ipv4.dstAddr : lpm;
    }
    actions {
        on_miss;
        fib_hit_nexthop;
    }
    size : IPV4_LPM_TABLE_SIZE;
}
```

```
control ingress {
    if (valid(ipv4)) {
        apply(port_mapping);
        apply(bd);
        apply(ipv4_fib) {
            on_miss {
                apply(ipv4_fib_lpm);
            }
        }
        apply(nexthop);
    }
}
```

```
action fib_hit_nexthop(nexthop_index) {
    modify_field(ingress_metadata.nexthop_index, nexthop_index);
    subtract_from_field(ipv4.ttl, 1);
}
```

Table Types

- Exact: value == table entry
  - E.g. IPv4 host route
- Ternary: value AND mask == table entry
  - Wildcard
- LPM: Longest Prefix Match
  - Special case of ternary (1111….11110000…..0000)
- Range: low entry <= value <= high entry
- Valid: table entry = {true, false}
  - True: header field is valid
  - False: header field is not valid

P4 Checksum Support

```
header ipv4_t ipv4;

field_list ipv4_checksum_list {
        ipv4.version;
        ipv4.ihl;
        ipv4.diffserv;
        ipv4.totalLen;
        ipv4.identification;
        ipv4.flags;
        ipv4.fragOffset;
        ipv4.ttl;
        ipv4.protocol;
        ipv4.srcAddr;
        ipv4.dstAddr;
}
```

```
field_list_calculation ipv4_checksum {
    input {
        ipv4_checksum_list;
    }
    algorithm : csum16;
    output_width : 16;
}

calculated_field ipv4.hdrChecksum  {
    verify ipv4_checksum;
    update ipv4_checksum;
}
```

```
header_type ipv4_t {
    fields {
        version : 4;
        ihl : 4;
        diffserv : 8;
        totalLen : 16;
        identification : 16;
        flags : 3;
        fragOffset : 13;
        ttl : 8;
        protocol : 8;
        hdrChecksum : 16;
        srcAddr : 32;
        dstAddr: 32;
    }
}
```

Checksum Algorithms

- XOR16
- CSUM16
- CRC16
- CRC32
- Programmable_CRC
  - Arbitrary CRC polynomial

Additional P4 Features

- Counters
  - Type: bytes or packets
  - Min-width
  - Saturating: stop counting; default is wrap
- Meters
- Registers
- Resubmit (original packet + metadata)
- Recirculate (packet after egress modifications)

P4 Control Flow

- If/else
- +, *, -, <<, >>, &, |, ^
- ~, -
- OR, AND
- >, >=, ==, <=, <, !=

Work Flow

- Write P4 program, typically these source files:
  - foo.p4
  - headers.p4
  - parser.p4
- Convert P4 program to JSON configuration
- Run P4 software switch with JSON config

More information

www.p4.org

Thank You

Ronald van der Pol

SURFnet

Ronald.vanderPol@rvdp.org