User-Programmable Software Switches

Nick McKeown
Experience so far
Experience with P4 programs written for Tofino @ 6.5Tb/s
PISA: Protocol Independent Switch Architecture
PISA: Protocol Independent Switch Architecture
P4 program

P4 Compiler

PISA Programmable Switch
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**

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.

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.

Table 1: Fields

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Header Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Dec 2009</td>
<td>12 fields (Ethernet, TCP/IP)</td>
</tr>
<tr>
<td>1.1</td>
<td>Feb 2011</td>
<td>15 fields (MPLS, Ethernet, UDP)</td>
</tr>
<tr>
<td>1.2</td>
<td>Dec 2011</td>
<td>36 fields (MPLS, Ethernet, metadata)</td>
</tr>
<tr>
<td>1.3</td>
<td>Jun 2012</td>
<td>40 fields (ARP, ICMP, HTTP, etc.)</td>
</tr>
<tr>
<td>1.4</td>
<td>Jul 2013</td>
<td>41 fields</td>
</tr>
</tbody>
</table>

Figure 1: P4 is a language to configure switches.
P4 Program Sections

**program.p4**

**Data Declarations**

```p4
data_type ethernet_t { ... }
data_type l2_metadata_t { ... }

header ethernet_t ethernet;
header vlan_tag_t vlan_tag[2];
metadata l2 metadata_t l2 meta;
```

**Parser Program**

```p4
class parser parse_ethernet {
    extract(ethernet);
    return switch(ethernet.ethertype) {
        0x8100: parse_vlan_tag;
        0x0800: parse_ipv4;
        0x8847: parse_mpls;
        default: ingress;
    }
}
```

**Table + Control Flow Program**

```p4
table port_table { ... }
control ingress {
    apply(port_table);
    if (l2_meta.vlan_tags == 0) {
        process_assign_vlan();
    }
}
```

P4 program defines what each table CAN do
Control Plane Roles

P4 defined what each table CAN do

Control plane or NOS decides **what the switch actually does**
Protocols and table complexity 20 years ago
Datacenter ToR today
public switch.p4
Visibility and Measurement
Natural questions

• Which switches did it visit to get here?
• What rules did it match in each switch?
• What version of the switch rule tables were present?

• Which queue did each switch put our packet in?
• What was the precise queue occupancy when my packet arrived?
• How long did it wait?
• Whose packets did it share a queue with?
Two approaches
Each is a P4 program

1. Packet postcards
   • Switch generates a small time-stamped digest for every packet
   • Sends to server(s) for logging and processing
   • **Pros**: Can replay network history. Packet sizes unchanged.
   • **Cons**: Lots of extra traffic.
Packet Postcards
Two approaches  
Each is a P4 program  

1. Packet postcards  
   • Switch generates a small time-stamped digest of every packet header and table version  
   • Sends to server(s) for logging and processing  
   • **Pros**: Can replay network history. Packet sizes unchanged.  
   • **Cons**: Lots of extra traffic.  

2. Inband Network telemetry (INT)  
   • Data packets carry instructions to insert state into packet header  
   • **Pros**: No additional packets. Can replay network history.  
   • **Cons**: Packet size increases.
In-band Network Telemetry (INT)

“Insert: switchID, time, matched rules, queue occupancy, switch metadata, ..., ..., ...”
```
INT.p4

```table int_table {
  reads {
    ip.protocol;
  }
  actions {
    export_queue_latency;
  }
}
```

```action export_queue_latency (sw_id) {
  add_header(int_header);
  modify_field(int_header.kind, TCP_OPTION_INT);
  modify_field(int_header.len, TCP_OPTION_INT_LEN);
  modify_field(int_header.sw_id, sw_id);
  modify_field(int_header.q_latency, intrinsic_metadata.deq_timedelta);
  add_to_field(tcp.dataOffset, 2);
  add_to_field(ipv4.totalLen, 8);
  subtract_from_field(ingress_metadata.tcpLength, 12);
}
```

**Example:** Add switch ID and queue latency to packet
PLT: Path and latency tracking in data-plane

How does it work?
- Collect physical path and hop latency of every packet via INT
- Last hop creates a record per connection
- Records any sudden change in path or latency

How is it used?
- Quickly detect changes in path-latency at line-rate, in data-plane
- Confirm routing table or ACL rule changes in real time
- Identify connections affected by failure, recovery or maintenance events
CT: Congestion tracking in data-plane

How does it work?

- During congestion, switch takes “snapshot” of every packet
- Snapshot contains packet ID and packet metadata for analysis

How is it used?

- Detect congestion incidents and identify events leading to congestion
- Identify culprit that is causing queue builds-up
- Identify persistent congestion and transient congestion
L4LB: Add L4 load balancing to every switch

How does it work?

- Ensure **per-connection consistency**: Forward every packet in a connection to the same DIP
- Switch maintains per-connection state (typically five million or more)

How is it used?

- Cost saving: Eliminate thousands of servers

P4 prototype available from demo at the 2\textsuperscript{nd} P4 workshop
Custom traffic monitoring and filtering

**General-purpose stateful memory & Custom hashing**
→ Explosion of probabilistic traffic monitoring and filtering schemes

**Bloom-filter-based whitelist**
- For example, remember $O(10^7)$ items with very low false positives

**Heavy-hitter detection via count-min sketch**
- For example, track the frequency of $O(10^7)$ items

- Switches encode flow-sets using Invertible Bloom Filter and export the encodings frequently to monitoring servers -- once every few msec
- Monitors decode the encodings network wide and produce NetFlow-like records
Dynamic source routing

Forward packets/flowlets/flows based on current path conditions
- Path condition: Link utilization, queue depth, hop latency, end-to-end latency, etc.

“HULA” at SOSR’16
Scalable high-frequency OAM

- Offload BFD entirely to data plane using programmable packet generator + stateful memory
- Switches maintain many thousands of BFD sessions with msec-level hello frequency
Various types of congestion control

Explicit congestion-control protocols running in switches
  • RCP, XCP, TeXCP, etc.

Hybrid congestion control – or “Timely++”
  • Switches insert ID and queuing latency in every packet
  • Sender decides best rate for each connection

Host-to-dst-ToR admission control (network-level VoQ)
  • Last-hop ToR enforces “hose-model” traffic via admission control
  • High throughput, low latency, and (nearly) lossless without pausing
  • Enhanced: hosts expose more info to network, such as traffic type, message size, deadline, etc.
Flowlet Switching

Flowlet Table (Register Array)

<table>
<thead>
<tr>
<th>last time</th>
<th>flowlet id</th>
</tr>
</thead>
<tbody>
<tr>
<td>82019445</td>
<td>4</td>
</tr>
<tr>
<td>82028039</td>
<td>13</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>81084924</td>
<td>29</td>
</tr>
<tr>
<td>82148703</td>
<td>7</td>
</tr>
</tbody>
</table>

if (current_time − last_time > timeout) {
    flowlet_id += 1;
}
last_time = current_time;

Hash (e.g., 5-tuple)

ECMP and LAG
Next hop and port selection using
hash of 6-tuple: 5-tuple + flowlet id

packet hdr & metadata
Flowlet Switching

### Flowlet Table
(Register Array)

<table>
<thead>
<tr>
<th>last time</th>
<th>flowlet id</th>
</tr>
</thead>
<tbody>
<tr>
<td>82019445</td>
<td>4</td>
</tr>
<tr>
<td>82028039</td>
<td>13</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>81084924</td>
<td>29</td>
</tr>
<tr>
<td>82148703</td>
<td>7</td>
</tr>
</tbody>
</table>

Hash(e.g., 5-tuple) 

packet hdr & metadata

/* data structure */

```c
register flowlet_lasttime {
  width : 32;
  instance_count : 8192;
}
```

```c
register flowlet_id {
  width : 16;
  instance_count : 8192;
}
```

**ECMP and LAG**
Next hop and port selection using hash of 6-tuple:
5-tuple + flowlet id
Flowlet Switching

**Flowlet Table**
(Register Array)

<table>
<thead>
<tr>
<th>last time</th>
<th>flowlet id</th>
</tr>
</thead>
<tbody>
<tr>
<td>82019445</td>
<td>4</td>
</tr>
<tr>
<td>82028039</td>
<td>13</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>81084924</td>
<td>29</td>
</tr>
<tr>
<td>82148703</td>
<td>7</td>
</tr>
</tbody>
</table>

Packet hdr & metadata

*Hash (e.g., 5-tuple)*

Hash function

```
#define FLOWLET_MAP_SIZE 13 // 8K table size
#define FLOWLET_INACTIVE_TOUT 50000 // 50ms

/* hash input fields */
field_list l3_hash_fields {
   // 5 tuple
}

/* hash function */
field_list_calculation flowlet_map_hash {
   input {
      l3_hash_fields;
   }
   algorithm : crc16;
   output_width : FLOWLET_MAP_SIZE;
}
```

ECMP and LAG

Next hop and port selection using hash of 6-tuple:

5-tuple + flowlet id
Flowlet Switching

Flowlet Table

<table>
<thead>
<tr>
<th>last time</th>
<th>flowlet id</th>
</tr>
</thead>
<tbody>
<tr>
<td>82019445</td>
<td>4</td>
</tr>
<tr>
<td>82028039</td>
<td>13</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>81084924</td>
<td>29</td>
</tr>
<tr>
<td>82148703</td>
<td>7</td>
</tr>
</tbody>
</table>

Hash(e.g., 5-tuple)

packet hdr & metadata

ECMP and LAG
Next hop and port selection using hash of 6-tuple: 5-tuple + flowlet id

```
table flowlet {
    actions { lookup_flowlet_map; }
}

control ingress {
    apply(flowlet);
    apply(ecmp_group);
    apply(ecmp_nhop);
    apply(forward);
}

action lookup_flowlet_map() {
    modify_field_with_hash_based_offset(ingress_metadata.flowlet_map_index, 0, flowlet_map_hash, FLOWLET_MAP_SIZE);

    ...
    add_to_field(ingress_metadata.flowlet_id, ingress_metadata.flow_ipg > FLOWLET_INACTIVE_TOUT ? 1 : 0)
    register_write(flowlet_id, ingress_metadata.flowlet_map_index, ingress_metadata.flowlet_id);
}
Heavy-Hitter Detection (HHD)

Heavy hitters (a.k.a elephant flows)
- A small number of flows (hundreds or thousands) contribute most network traffic
- Often transient, hard to proactively install counters
- Major source of network congestion
- Penalize delay-sensitive mice flows

Instant HHD in switch dataplane
- Detect every millisecond
- Useful in DC networks with small RTT and shallow buffer
- Counting, detection, reaction all at line-rate, in dataplane
Heavy-Hitter Detection with count-min sketch

Probabilistic data structure: counting Bloom filter

**Counting**
- Each flow computes multiple hash indices, adding packet size to the indexed locations of counter array
- Flows can hash-collide, adding to a common counter instance

```
f1
+100
0 100 0 100 0 300 0 0 200 200

f2
+200
```

**Detection**
- Take *minimum* of the counter values and compare to threshold

**Reaction**
- Dynamic de-prioritization, metering, etc
HHD.p4 (two hash-way example)

/* data structure */
register counter_array1 {
    width : 32;
    instance_count : 2048;
}
register counter_array2 {

/* hash input fields */
field_list l3_hash_fields {
    ipv4.srcAddr;
    ipv4.dstAddr;
    ipv4.protocol;
    tcp.srcPort;
    tcp.dstPort;
}

/* hash functions */
field_list_calculation hash1 {
    input { l3_hash_fields; }
    algorithm : crc16;
    output_width : 11; // 11=log2(2048)
}
field_list_calculation hash2 {

/* metadata variables */
header_type hhd_metadata_t {
    fields {
        index1 : 11;
        index2 : 11;
        count_val1 : 32;
        count_val2 : 32;
    }
}
metadata hhd_metadata_t md;

/* counting: counter read/update/write */
action count1() {
    /* compute hash index into md.index1 */
    modify_field_with_hash_based_offset(
        md.index1, 0, hash1, 11);
    register_read(md.count_val1, counter_array1, md.index1);
    add_to_field(md.count_val1, ipv4.len);
    register_write(counter_array1, md.index1, md.count_val1);
}
action count2() {
}
action count_all() {
    count1();
    count2();
}
HHD.p4

/* table to run action */
table counting_table {
  actions { count_all; }
  size : 1;
}

/* control function */
control ingress {
  apply(counting_table);

  /* detection & reaction */
  /* if every count_val is larger than threshold, de-prioritize */
  if (md.count_val1 > THRESHOLD and md.count_val2 > THRESHOLD) {
    apply(deprioritization_table);
  }
}
Key-Value Stores in P4

• **SwitchKV**: Key-value load-balancer and cache (e.g. for memcache) [NSDI 2016]

• **Paxos in P4**: Paxos leadership election algorithm [ACM CCR 2016]
User-programmable
Software Switches
A few choices

• Hand-coded C in user-space or kernel
• eBPF in kernel
• User space C with DPDK
• P4 compiled to user-space or kernel

Converged approach: P4-eBPF and eBPF-P4 cross compilers
PISCES: Protocol Independent Software Switch
Mohammad Shahbaz, Sean Choi, Jen Rexford, Nick Feamster, Ben Pfaff, NM
Sigcomm 2016

Problem: Adding new protocol feature to OVS is complicated
• Requires domain expertise in kernel programming and networking
• Many modules affected
• Long QA and deployment cycle: typically 9 months

Approach: Specify forwarding behavior in P4; compile to modify OVS

Question: How does the PISCES switch performance compare to OVS?
Native OVS expressed in P4
## Complexity Comparison

<table>
<thead>
<tr>
<th></th>
<th>LOC</th>
<th>Methods</th>
<th>Method Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native OVS</td>
<td>14,535</td>
<td>106</td>
<td>137.13</td>
</tr>
<tr>
<td>ovs.p4</td>
<td>341</td>
<td>40</td>
<td>8.53</td>
</tr>
</tbody>
</table>

- 40x reduction in LOC
- 20x reduction in method size

<table>
<thead>
<tr>
<th></th>
<th>Files Changed</th>
<th>Lines Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVS ovs.p4</td>
<td>28</td>
<td>411</td>
</tr>
<tr>
<td>Tunnel OAM Flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVS ovs.p4</td>
<td>18</td>
<td>170</td>
</tr>
<tr>
<td>TCP Flags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVS ovs.p4</td>
<td>20</td>
<td>370</td>
</tr>
</tbody>
</table>

- Code mastery no longer needed
User-programmable Software Switches

1. Open-source behavioral model and compiler at P4.org
2. OVS: Talk by Shahbaz later today...
3. VPP: Work in progress
How to learn more about P4
P4.org – P4 Language Consortium

BOARD MEMBERS
Three Board Members oversee the consortium:

Nick McKeown
Stanford University

Jennifer Rexford
Princeton University

Amin Vahdat
Google

P4 allows network engineers to change the way their switches process packets after they are deployed.
P4.org – P4 Language Consortium

- Developers Day on Wednesday at Stanford!
- Tutorials at conferences (e.g. Sigcomm, ONS)
- Annual P4 Workshop in May/June
- Boot camps for PhD students
- Open for free to any individual or organization
# P4.org Members

**Original P4 Paper Authors:**
- BAREFOOT NETWORKS
- Google
- intel
- Microsoft
- Princeton University
- Stanford University

## Operators/End Users
- Alibaba Group
- Baidu 百度
- FOX
- Goldman Sachs
- kt
- Microsoft
- Tencent 腾讯
- at&t
- COMCAST
- Google
- Intel
- SK telecom

## Systems
- BROCADE
- Cisco
- CORSA DAHC
- Dell
- Hewlett Packard Enterprise
- Huawei
- Inventec
- Juniper
- Netberg
- NoviFlow
- MoSys
- NETRONOME
- PLUMgrid
- VMware
- Xilinx

## Targets
- AEP NYX
- Atomic Rides
- BAREFOOT NETWORKS
- Broadcom
- Cavium
- centec networks
- EZChip
- Freescale
- Intel
- Marvell
- Mellanox Technologies

## Solutions/Services
- EstiNet
- happiest minds
- global itech
- SDNLAB
- Flow Research

## Academia/Research
- Bii
- Eötvös Loránd University
- National Chiao Tung University
- Princeton University
- salzburgresearch
- Stanford University
- University of Luxembourg
- University of São Paulo
- Virginia Tech
- Instituto Federal do Rio Grande do Sul
Five things on the horizon for P4.....
Separation of language from architecture
Reference architectures for portability
Extend P4 to express packet scheduling and QoS disciplines
Extend P4 to express stateful processing
5 Cross-compilers to-from BPF
A long-term aspiration

Declared network forwarding behavior

Automatically partition and generate code

P4 code

P4 code

P4 code

P4 code

P4 code

P4 code

P4 Compiler

OVS

NIC

P4 code

P4 code

P4 code

P4 code

P4 code

P4 code

NIC

OVS
Thank you