Outline

• Introduction to P4 and P4 Runtime
• P4 support in ONOS
• Future plans for Trellis
P4 and P4 Runtime Overview

Fixed-function data plane pipeline

Programmable (or fixed) data plane pipeline

Variations in data plane pipelines are hard to abstract

P4 enables custom pipelines to meet application needs

P4 Runtime allows custom pipelines to be loaded and controlled

A COMPLEX PIPELINE

THAT DOES EVERYTHING

...different vendors in their own way

Application

OpenFlow

Table management

Application

P4 program

P4 Runtime

gRPC

Table { match actions }

compile

P4 Runtime

Table management

Packets

Packets

JUST WHAT

I NEED
Programmable Switch Architecture

Programmer declares the headers that should be recognized and their order in the packet.

Programmer defines the tables (match type, actions) and the processing algorithm.

Programmable Match-Action Pipeline

Programmable ALUs with computational capabilities and stateful memories.

Slide courtesy: p4.org
P4 Packet Processing Language

- Domain-specific language to specify packet forwarding behaviours
- Open source consortium: P4.org
- Hardware agnostic, can be compiled to programmable ASICs, FPGAs, NPUs, etc.
- Value as a description language for fixed-function devices

```p4
header ethernet_t {
  bit<48> dst_addr;
  bit<48> src_addr;
  ...
}
header ipv4_t {
  bit<4> version;
  bit<4> ihl;
  bit<8> diffserv;
  ...
}

action set_next_hop(bit<48> dst_addr) {
  ethernet.dst_addr = dst_addr;
  ipv4.ttl = ipv4.ttl - 1;
}

parser parser_impl(packet_in pkt, out headers_t hdr)
{ /* Parser state machine */ }

table ip_table {
  key = { ipv4.dst_addr : LPM; }
  actions = { set_next_hop();
              drop(); send_to_ctrl(); }
  size = 4096;
}
```

Example P4 code
P4 Workflow

P4 Program → P4 Compiler → Configuration binary → Data Plane

Control Plane:
- Add/remove table entries
- Extern control
- Packet-in/out

Data Plane:
- Tables
- Extern objects

CPU port

P4-enabled switch

User supplied
Vendor supplied
**P4 Runtime**

- **Framework for runtime control of P4 targets**
  - Open-source API + server implementation
    - [https://github.com/p4lang/PI](https://github.com/p4lang/PI)
    - Initial contribution by Google and Barefoot Networks

- **Work-in-progress by the p4.org API WG**

- **Protobuf-based API + gRPC client/server impl.**
  - Many RPC features for free (e.g. authentication)

- **P4 program-independent**
  - API doesn’t change with the P4 program

- **Enables field-reconfigurability**
  - Ability to push new P4 program once switches have been deployed
P4 on ONOS
P4 on ONOS Today

• Applications can bring their own P4 pipelines

• **Northbound API to control any P4 program**
  • Ways to map existing protocol-dependent ONOS APIs to P4 pipelines
  • New pipeline-agnostic ONOS API to control custom pipelines

• **New P4 device drivers**
  • Barefoot Tofino-based switches
  • BMv2 software switch (great for prototyping)

• **P4 Runtime southbound interface**
  • Protocol support for P4 Runtime and gRPC
P4 on ONOS Demo

L123 SDN NFV World Congress 2017

Video!
https://youtu.be/BE_y-Sz0WnQ
Takeaways so far

• **P4 offers a formal contract between controller and switch**
  • Controller’s view of the pipeline (P4 program) is implemented by the switch
  • P4 Runtime API allows to control **any** pipeline → silicon independence!
    • No need to extend the API
    • API is pipeline-agnostic by definition

• **P4 Runtime offers value for fixed-function devices**
  • Provided that their behavior can be expressed in P4
  • Or, provided a compiler to map the P4 logical pipeline to the physical one
What’s next for CORD:
fabric.p4
fabric.p4: P4-based CORD Fabric

● **Goal:** bring more heterogeneity in the CORD fabric with P4 silicon
  ○ e.g. Barefoot Tofino, or any other vendor that offers a P4 compiler

● **Short-term scope - P4-based underlay (Spring 2018)**
  ○ Design a P4 pipeline (fabric.p4) that is equivalent to the OF-DPA one
  ○ Use fabric.p4 as a drop-in replacement for the current Trellis underlay
    ▪ **Do not change** the ONOS application programming the pipeline

● **Long-term - offload x86 processing to fabric**
  ○ P4-based overlay, i.e. move VXLAN handling from OVS to the ASIC
  ○ CORD VNFs offloading (will come to this later)
fabric.p4: where we are today

- Prototype P4 code and ONOS driver for fabric.p4 (Pipelinier)
  - Under onos/pipelines/fabric
VNF offloading

• Programmable data planes offer great degree of flexibility beyond plumbing

<table>
<thead>
<tr>
<th>Progr. ASIC capabilities</th>
<th>VNF building blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary header parsing/deparsing</td>
<td>Domain specific encap/decap (e.g. PPPoE termination, GTP, etc.)</td>
</tr>
<tr>
<td>Stateful memories</td>
<td>TCP connection tracking (L4 load balancing, NAT, firewall, etc.)</td>
</tr>
<tr>
<td>Computational capabilities</td>
<td>Billing</td>
</tr>
</tbody>
</table>

• Benefits
  • Scalability - VNFs executed at wire speed
  • Low latency and jitter - avoid non-determinism of x86 processing
Thanks!

Join the P4 Brigade!
P4 support in ONOS

**Pipeline-agnostic applications**

- **PD APIs**
  - Flow Rule
  - Flow Objectives
  - Intents

**Pipeline-aware application**

- **PI APIs**
  - Pipeline-specific entities

**PD-to-PI translation serv.**

- Flow rule, groups, etc.

**PI Framework**

- **Pipeconf (oar)**

**Core**

- Loadable Pipeline Configuration containing
  - Pipeline model
  - Target-specific artifacts compiled from custom P4 program (P4info, BMv2 JSON, Tofino bin, etc.)
  - Flow rule interpreter (Java code for PD-to-PI translation)

**Driver**

- Other drivers
- Tofino
- BMv2
- P4Runtime
- gNMI

**Protocol**

- gRPC
- Other protocols

**Target-specific artifacts from the Pipeline Configuration are used to initialize the switch pipeline.**

**Pipeline Configuration**

- **PD APIs**
  - Flow Rule
  - Flow Objectives
  - Intents

- **PI APIs**
  - Pipeline-specific entities

**Future Work**

- Other drivers
- Other protocols

**Default drivers**

- P4Runtime
- gNMI
p4runtime.proto simplified excerpts:

```protobuf
message TableEntry {
  uint32 table_id;
  repeated FieldMatch match;
  Action action;
  int32 priority;
  ...
}

message FieldMatch {
  uint32 field_id;
  message Exact {
    bytes value;
  }
  message Ternary {
    bytes value;
    bytes mask;
  }
  ...
  oneof field_match_type {
    Exact exact;
    Ternary ternary;
    ...
  }
}

message Action {
  uint32 action_id;
  message Param {
    uint32 param_id;
    bytes value;
  }
  repeated Param params;
}
```

To add a table entry, the control plane needs to know:

- IDs of P4 entities
  - Tables, field matches, actions, params, etc.
- Which field matches are defined in which table
  - The match type, bitwidth, etc.
- Which parameters are required by which actions
- Other P4 program attributes

Full protobuf definition: