P4 support in ONOS

Carmelo Cascone
ONF
Agenda

● Introduction to P4 and P4Runtime
● P4 support in ONOS
  ○ Architecture
  ○ PI Framework
  ○ Workflow
● Demo!
P4, P4Runtime, P4Info

Concepts and workflow
P4

- Open-source domain-specific language (DSL) for programmable dataplanes
  - Protocol independent
  - Support many targets: ASICs, FPGAs, NPUs, software switches
  - Field reconfigurable: add/remove capabilities after switches are deployed

- Based on the well known match+action forwarding model

- Many benefits
  - Add new protocols, remove unused ones
  - Telemetry
  - More advanced capabilities (stateful processing, VNF offloading, etc.)
P4, the ONOS developer perspective...

Fixed-function dataplane

- Simple forwarding app
- Hard to code, debug, manage!
- Table mgmt
- A COMPLEX PIPELINE
  THAT DOES EVERYTHING ...
  POORLY

Programmable dataplane

- Simple forwarding app
- Easy to code debug, manage!
- Table mgmt
- P4 program
- Table { match actions }
  my_pipeline.p4
- ONOS
- JUST WHAT
  NEED
  I

Complex pipeline that does everything poorly.

ONOS

Simple forwarding app

Table mgmt

P4 program
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.
- Programmer declares how the output packet will look on the wire.

Programmable Parser

Programmable Match-Action Pipeline

Programmable Deparser

Slide courtesy: Vladimir Gurevich, P4_16 Tutorial. P4 Workshop 2017
P4 workflow

- **P4 workflow**
  - **Data plane**
    - my_program.p4
    - arch.p4
  - **P4 Compiler**
    - Target-specific configuration/binary
  - **P4 Runtime**
    - Tables
    - Extern objects
  - **Control plane**
    - Manufacturer supplied
  - **BMv2**

- **P4 Compiler**
  - **p4c**

P4Runtime

- Framework for **runtime control** of P4 devices
  - Open-source: [https://github.com/p4lang/PI](https://github.com/p4lang/PI)
- Developed by the **p4.org API WG**
- Targeted for **remote controllers**
  - Protobuf + gRPC implementation
- **P4 program-independent**
  - API doesn’t change with the P4 program
- Enables **field-reconfigurability**
  - Ability to push new P4 program to the device
p4runtime.proto

- Defines interaction with entities defined in a P4 program
  - Tables, counters, meters, externs, etc.
- Bidirectional stream channel
  - For packet-ins/outs, mastership updates, etc.

How to get such IDs?
p4c workflow

- **P4Info.proto**
  - Captures target-independent P4 program attributes
  - Defines integer IDs for P4 entities, etc.

- **Target config**
  - P4Info + device-specific configuration (e.g. BMv2’s JSON, Tofino binary, etc.)

Slide courtesy: S. Abdi, W. Mohsin, Y. Yetim, A. Ghaffarkhah.
P4 Program-Dependent Controller Interface for SDN Applications. P4 Workshop 2017
P4Info example

action set_vrf(bit<32> id) {
   meta.vrf_id = id;
}

table vrf_classifier_table {
   key = {
      hdr.ethernet.etherType : exact;
      hdr.ethernet.srcAddr : ternary;
      smeta.ingress_port: exact;
   }
   actions = {
      set_vrf;
   }
   default_action = set_vrf(0);
}

Slide courtesy: S. Abdi, W. Mohsin, Y. Yetim, A. Ghaffarkhah.
P4 Program-Dependent Controller Interface for SDN Applications. P4 Workshop 2017
P4Runtime example

```c
action set_vrf(bit<32> id) {
  meta.vrf_id = id;
}
table vrf_classifier_table {
  key = {
    hdr.ethernet.etherType : exact;
    hdr.ethernet.srcAddr : ternary;
    smeta.ingress_port: exact;
  }
  actions = {
    set_vrf;
  }
  default_action = set_vrf(0);
}
```
P4 support in ONOS
Challenge

● How can we control and configure P4-enabled devices?
● ONOS initially designed around OpenFlow fixed-function dataplane
  ○ NB abstractions morphed around OpenFlow (e.g. same match/actions)
  ○ Immutable pipeline
● With P4…
  ○ Generalized forwarding abstraction (e.g. arbitrary match/actions)
  ○ Mutable pipeline (devices can support different pipelines in time)
Architecture overview

2 modes of operations:
- Standard match, actions (i.e. OpenFlow ones)
- Pipeline-specific ones

Pack together everything needed to:
- Understand a P4 program/pipeline
- Control that pipeline (driver)
- Deploy P4 program to device
PI Framework

- PI = protocol/program/pipeline independent
- Classes, services, and driver behaviours to model and control programmable data planes
  - Classes starting with Pi*, e.g. PiPipeconf, PiTableEntry, etc.
- Modelled around P4 and PSA
  - Define table entries, counters, etc.
- @beta: expect changes
PI Pipeconf

- Pack together data and code necessary to let ONOS:
  - Understand, control, and deploy an arbitrary pipeline
- Provided to ONOS as an application (.oar)

1. **Pipeline model**
   - Pipeline entities description (i.e. parsed P4 program)
2. **Driver behaviors** (pipeline-specific)
   - E.g. FlowObjective’s Pipeliner
3. **Target-specific extensions**
   - E.g. BMv2 JSON, Tofino binary
   - P4Info, needed for P4Runtime’s integer ID-name mapping

Pipeconf registered via **PipeconfService** → available to all ONOS components
Device discovery

**Device ID**: bmv2:1

**P4Runtime**
- Server addr: 192.168.56.1
- Port: 5001

**Pipeconf ID**: my-pipeconf

**Driver**: bmv2

**Extensions**: BMV2_JSON, P4INFO

**netcfg.json**

**BMv2 PI Pipeline Programmable**
- Set pipeline config (BMv2 JSON + P4Info)

**BMv2 Device Handshaker**
- Open TCP socket to gRPC server
Flow operations

Pipeconf-based 3 phase translation:
- Flow Objective → Flow Rule
  ○ Via Pipeliner
- Flow Rule → PI Table Entry
  ○ Via PI Pipeline Interpreter
- PI Table Entry to P4Runtime msg
  ○ Via P4Info
PI Pipeline Interpreter

- Expose methods to let ONOS “understand” a given P4 program
  - Maps protocol-dependent ONOS constructs to PI entities
- Pipeline-specific driver behaviour (included in the pipeconf)
  - P4 program writers need to provide implementation
- Example: flow rule translation
  - Match
    - 1:1 mapping between ONOS criteria and P4 header names
  - Action
    - Problem: P4 allows only one action per table entry, ONOS many
    - E.g. header rewrite + output: 1 action with 2 parameters in P4, 2 actions in ONOS
    - How to map many actions to one? Need interpretation logic (i.e. Java code)!
- Used also for other purposes
  - Map ONOS table integer IDs to names, packet I/O operations, table counters, etc.
Pipeline-aware flow rules

- **Match**
  - ONOS standard criteria → provided mapping via interpreter
  - PI Criterion → collection of field matches, each one using:
    - P4 header field name (e.g. my_tunnel_header.tunnel_id)
    - Match type (e.g. exact, ternary, LPM)
    - Value/mask (bytes)

- **Action**
  - ONOS standard instructions → provided mapping via interpreter
  - PI Instruction → specify action using
    - P4 action name (e.g. set_agress_port)
    - Collection of action parameters
      - Parameter name (e.g. port_id)
      - Parameter value (bytes)

Standard/PI criteria/instructions can be combined together
Packet I/O operations

- **Packet-in**: packet received at a switch port encapsulated and sent to the controller
- **Packet-out**: packet generated at the controller sent through a switch port
- With P4, encapsulation format defined by programmer → Need interpreter!

Example:
```
@controller_header("packet_in")
header packet_in_header_t {
    bit<9> ingress_port;
}
@controller_header("packet_out")
header packet_out_header_t {
    bit<9> egress_port;
}
```

Diagram:
- App
- Packet Request/Manager Serv.
- P4Runtime Packet Provider
- PI Pipeline Interpreter
- BMv2 Packet Programmable
- PI Packet Operation
- P4RuntimeClient
- P4Info
- ONOS Core
- Device Driver
- Pipeconf
- P4Runtime protobuf msg
ONOS+P4 workflow recap

● **Write P4 program**
  ○ If you need, define SDN-like behaviours (packet-in/out headers, actions)

● **Compile it**

● **Assemble Pipeconf**
  ○ Pipeline model (e.g. BMv2 JSON)
  ○ Pipeline-specific driver behaviours:
    ■ Interpreter
    ■ Pipeliner (if you need Flow Objectives)
    ■ Any other behaviour that depends on the pipeline
  ○ Target-specific extensions
    ■ P4Info, BMv2 JSON, Tofino binary, etc.

● **Write your own pipeline-aware application or use existing pipeline-agnostic ones**

● **Enjoy!**
Demo!
Takeaway

Today ONOS offers capabilities to:

- **Store** and **deploy** P4-defined pipelines (PI Pipeconf Service)
- **Control** pipeline entities (standard ONOS NB APIs)
  - Table entries, counters, packets I/O, etc.
- Support for **pipeline-agnostic** applications
  - Provided mapping via Interpreter and Pipeliner
- Support **pipeline-aware** applications (PI Framework)
- Integrate P4 devices into **heterogenous networks**
  - Control P4, OpenFlow, or Netconf devices through the same high-level APIs

While maintaining **high availability, scalability and performance** key characteristics to the ONOS platform
Thanks!
Evolving control and configuration
gNMI: configuration

- RPCs and behaviors for managing state on a device
  - set/get config, retrieve capabilities, subscribe to notifications
- supports state retrieval (via streaming telemetry or snapshots)
- built on the open source gRPC framework (gRPC ⊂ gNMI)
- gNMI defines a gRPC service using protobuf IDL designed to carry any tree-structured data (not limited to YANG-modeled data)
  - addressable via paths
  - has well-defined serialization
gNMI: configuration

- Port description, port statistics, manage LEDs, etc.
- P4.org API WG suggests using existing OpenConfig Yang based data models → **not reinventing the wheel**
  - Currently OpenConfig Yang models are supported in ONOS
- gNMI → support from BMv2, PI (switch-side server for runtime control)