Performance evaluation of ONOS support for P4Runtime
Proposed activity for the Sec&Perf brigade

Carmelo Cascone
MTS, ONF

June 17, 2019
ONF Sec&Perf Workshop @ TMA 2019
Outline

• P4Runtime recap
• P4Runtime subsystem in ONOS
• Performance considerations
P4 - Data plane pipeline programming language

mypipeline.p4

Compiler (provided by switch vendor)

Pipeline of match-action tables, programmable or fixed-function

Packets

ASIC, FPGA, NPU, or CPU
P4 workflow summary

User supplied

Vendor supplied

P4 Program

P4 Architecture Model

P4 Compiler

Target-specific configuration binary

Load

Tables

Extern objects

Control Plane

Add/remove table entries

Extern control

Packet-in/out

CPU port

Data Plane

P4Runtime

Slide courtesy P4.org
P4Runtime v1.0

- Released on Jan 2019
- Open source specification
  - Started by Google and Barefoot in mid-2016
  - Contributions by many industry professionals
- Based on continuous implementation feedbacks from Google and ONF
  - First ONF demo in Oct 2017

https://p4.org/p4-spec/
https://github.com/p4lang/p4runtime
P4Runtime overview

- **Protobuf-based API definition**
  - Efficient wire format
  - Automatically generate code to serialize/deserialize messages for many languages

- **gRPC-based transport**
  - Automatically generate high-performance client/server stubs in many languages
  - Pluggable authentication and security
  - Bi-directional stream channels

- **P4-program independent**
  - Allow pushing new P4 programs to reconfigure the pipeline at runtime

- **Equally good for remote or local control plane**
  - With or without gRPC
P4Runtime main features

- **Batched read/write of pipeline state**
  - Table entries, action groups, counters, registers, etc.

- **More robust mastership handling wrt OpenFlow**
  - With ordering of writes

- **Multiple master controllers via role partitioning**
  - E.g. local control plane for L2, remote one for L3

- **More flexible and efficient packet I/O**
  - OpenFlow-like packet-in/out with arbitrary metadata
  - Digests, i.e. batched notification to controller with subset of packet headers

- **Designed around PSA architecture**
  - But can be extended to others via Protobuf “Any” messages
P4 compiler outputs

1. Target-specific binaries
   ○ Used to realize switch pipeline
     (e.g. binary config for ASIC, bitstream for FPGA, etc.)

2. P4Info file
   ○ “Schema” of pipeline for runtime control
     ■ Captures P4 program attributes such as tables, actions, parameters, etc.
   ○ Protobuf-based format
   ○ Target-independent compiler output
     ■ Same P4Info for SW switch, ASIC, etc.

Full P4Info protobuf specification:
https://github.com/p4lang/p4runtime/blob/master/proto/p4/config/v1/p4info.proto
**P4Info example**

### basic_router.p4

```p4
...  

action ipv4_forward(bit<48> dstAddr,  
                   bit<9> port) {
    eth.dstAddr = dstAddr;
    metadata.egress_spec = port;
    ipv4.ttl = ipv4.ttl - 1;
}  
...

table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        ...
    }
    ...
}
```

### basic_router.p4info

```p4info
actions {
    id: 16786453
    name: "ipv4_forward"
    params {
        id: 1
        name: "dstAddr"
        bitwidth: 48
        ...
        id: 2
        name: "port"
        bitwidth: 9
    }
}
...

tables {
    id: 33581985
    name: "ipv4_lpm"
    match_fields {
        id: 1
        name: "hdr.ipv4.dstAddr"
        bitwidth: 32
        match_type: LPM
    }
    action_ref_id: 16786453
}
```
P4Runtime table entry WriteRequest example

```
basic_router.p4

action ipv4_forward(bit<48> dstAddr, bit<9> port) {
    /* Action implementation */
}
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        ...
    }
    ...
}
```

Logical view of table entry

```
hdr.ipv4.dstAddr=10.0.1.1/32
-> ipv4_forward(00:00:00:00:00:10, 7)
```

Control plane generates

```
WriteRequest message (protobuf text format)

device_id: 1
election_id { ... }
updates {
    type: INSERT
    entity {
        table_entry {
            table_id: 33581985
            match {
                field_id: 1
                lpm {
                    value: "\n\000\001\001"
                    prefix_len: 32
                }
            }
            action {
                action_id: 16786453
                params {
                    param_id: 1
                    value: "\000\000\000\000\000\000\000\000\000\n"
                }
                params {
                    param_id: 2
                    value: "\000\007"
                }
            }
        }
    }
}
```

Copyright © 2019 - Open Networking Foundation
An aside on Stratum and fixed-function switches...
Role of P4 for fixed-function chips

- P4 program tailored to apps / role - does not describe the hardware
- Switch maps program to fixed-function ASIC
- Enables portability
Project Stratum (ONF)

- **Vendor-agnostic implementation of next-gen SDN interfaces**
  - P4Runtime, gNMI (config), and gNOI (operations)
  - Production-ready, minimize bugs and improves time to market for vendor implementations

- **P4 compiler backend for fixed pipeline model (FPM)**
  - Produces mapping between P4-defined tables and SDK tables/APIs
  - Initial support for Broadcom Tomahawk and SDKLT

- **Extensive conformance test framework**
  - Along with a repository of tests
  - Make sure that vendor-specific pieces are implemented as expected
Stratum High-level Architectural Components

- P4Runtime
- gNMI
- gNOI
- Switch Broker Interface
- Table Manager
- Node/Chip Manager
- Chassis Manager
- Platform Manager
- Chip Abstraction Managers (E.g. ACL, L2, L3, Packet I/O, Tunnel)
- Switch SDK
- Platform API
- Switch Chip Drivers
- Platform Drivers
- Switch Chip(s)
- Peripheral(s)

Current support for:
- Barefoot Tofino
- BRCM Tomahawk
- BMv2 sw switch

Common (HW agnostic)
- Chip specific
- Platform specific
- Chip and Platform specific

Copyright © 2019 - Open Networking Foundation
ONOS support for P4Runtime
Design goals

ONOS originally designed to work with OpenFlow and fixed-function switches

Extended it to:

1. Allow ONOS users to bring their own P4 program
2. Allow existing apps to control any P4 pipeline without changing the app
   - i.e. provide app portability across many P4 pipelines
   - Re-use Trellis apps with other P4-capable switches
3. Allow new apps to control custom P4-defined protocols
   - e.g. apps for BNG and 4G/5G S/PGW control plane
Pipeconf - Bring your own pipeline!

- Package together everything necessary to let ONOS understand, control, and deploy an arbitrary pipeline
- Provided to ONOS as an app
  - Can use .oar binary format for distribution

1. Pipeline model
   - Description of the pipeline understood by ONOS
   - Automatically derived from P4Info

2. Target-specific binaries to deploy pipeline to device
   - E.g. BMv2 JSON, Tofino binary, FPGA bitstream, etc.

3. Pipeline-specific driver behaviors
   - E.g. mapping of ONOS flow programming API to P4 pipeline

Copyright © 2019 - Open Networking Foundation
Pipeconf support in ONOS

- **Pipeline-agnostic App**
  - FlowObjectives
  - Events (packet, topology, etc.)

- **Pipeline-aware App**
  - Pipeline-specific FlowRules, Groups, Meters, etc.

**ONOS**

**Core**

- Translation services
  - Uses pipeconf drivers

- PI Framework

- Pipeconf Store

**Drivers**

- Stratum

**Protocol**

- P4Runtime
- gNMI
- gNOI

- gRPC
  - Deploy P4 program, table management, config, operations

**Device with Stratum** (BF Tofino, BRCM Tomahawk, BMv2, etc.)
PI framework (@beta)

- **PI** = (data plane) protocol-independent
- **Model**: abstraction derived from P4Info
- **Runtime**: abstraction derived from P4Runtime
- **Service**: to operate on PI-capable devices

---

**onos/core/api/.../pi/model**
- DefaultPiPipeconf.java
- PiActionId.java
- PiActionModel.java
- PiActionParamId.java
- PiActionProfileId.java
- PiActionProfileModel.java
- PiControlMetadataId.java
- PiControlMetadataModel.java
- PiCounterId.java
- PiCounterModel.java
- PiDateTime.java
- PiDateTimeModel.java
- PiMatchFieldId.java
- PiMatchFieldModel.java
- PiMatchType.java
- PiMeterId.java
- PiMeterModel.java
- PiMeterType.java
- ...

**onos/core/api/.../pi/runtime**
- PiAction.java
- PiActionGroup.java
- PiActionGroupHandle.java
- PiActionGroupId.java
- PiActionGroupMember.java
- PiActionGroupMemberHandle.java
- PiActionParamId.java
- PiActionParamModel.java
- PiControlMetadata.java
- PiCounterCell.java
- PiCounterCellData.java
- PiCounterCellId.java
- PiEntity.java
- PiEntityType.java
- PiExactFieldMatch.java
- PiFieldMatch.java
- PiGroupKey.java
- PiHandle.java
- PiLpmFieldMatch.java
- ...

**onos/core/api/.../pi/service**
- PiFlowRuleTranslationStore.java
- PiFlowRuleTranslator.java
- PiGroupTranslationStore.java
- PiGroupTranslator.java
- PiMeterTranslationStore.java
- PiMeterTranslator.java
- PiMulticastGroupTranslationStore.java
- PiMulticastGroupTranslator.java
- PiPipeconfConfig.java
- PiPipeconfDeviceMappingEvent.java
- PiPipeconfMappingStore.java
- PiPipeconfMappingStoreDelegate.java
- PiPipeconfService.java
- PiPipeconfWatchdogEvent.java
- PiPipeconfWatchdogListener.java
- PiPipeconfWatchdogService.java
- PiTranslatable.java
- PiTranslatedEntity.java
- PiTranslationEvent.java
- ...

Translation Service

- Core service
- Translate pipeline-specific entities from protocol-dependent representations to PI ones
  - E.g. OpenFlow-like headers/criteria and actions to P4-specific ones

Diagram:
- Pipeconf
  - Translation service with validation (based on P4Info-derived pipeline model)
  - FlowRule
  - Group
  - Meter
  - PiTableEntry
  - PiActionGroup
  - PiMulticastGroupEntry
  - PiMeterCellConfig
Flow operations

Pipeconf-based 3-phase translation:

1. Flow Objective → Flow Rule
   - Maps 1 flow objective to many flow rules

2. Flow Rule → Table entry
   - Maps standard headers/actions to P4-defined ones
     E.g. ETH_DST→“hdr.ethernet.dst_addr”

3. Table Entry → P4Runtime message
   - Maps P4 names to P4Info numeric IDs
     “hdr.ethernet.dst_addr” → 3498746
PipelineInterpreter (driver behavior)

● Provide mapping from OpenFlow-derived ONOS headers/actions to P4 program-specific entities

● Example: flow rule translation
  ○ Match
    ■ 1:1 mapping between ONOS known headers and P4 header names
    ■ E.g. ETH_DST → ethernet.dst_addr (name defined in P4 program)
  ○ Action
    ■ ONOS defines standard actions as in OpenFlow (output, set field, etc.)
    ■ P4 allows only one action per table entry, ONOS many (as in OpenFlow)
    ■ E.g. header rewrite + output: 2 actions in ONOS, 1 action with 2 parameters in P4
    ■ How to map many actions to one? Need interpretation logic (i.e. Java code)!
## P4Runtime support in ONOS 2.1

<table>
<thead>
<tr>
<th>P4Runtime control entity</th>
<th>ONOS API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table entry</td>
<td>Flow Rule Service, Flow Objective Service</td>
</tr>
<tr>
<td></td>
<td>Intent Service</td>
</tr>
<tr>
<td>Packet-in/out</td>
<td>Packet Service</td>
</tr>
<tr>
<td>Action profile group/members, PRE multicast groups, clone sessions</td>
<td>Group Service</td>
</tr>
<tr>
<td>Meter</td>
<td>Meter Service (indirect meters only)</td>
</tr>
<tr>
<td>Counters</td>
<td>Flow Rule Service (direct counters)</td>
</tr>
<tr>
<td></td>
<td>P4Runtime Client (indirect counters)</td>
</tr>
<tr>
<td>Pipeline Config</td>
<td>Pipeconf</td>
</tr>
</tbody>
</table>

**Unsupported features - community help needed!**
Parser value sets, registers, digests
ONOS+P4 workflow recap

- **Write P4 program and compile it**
  - Obtain P4Info and target-specific binaries to deploy on device

- **Create pipeconf**
  - Implement pipeline-specific driver behaviours (Java):
    - Pipeliner (optional - if you need FlowObjective mapping)
    - Pipeline Interpreter (to map ONOS known headers/actions to P4 program ones)
    - Other driver behaviors that depend on pipeline

- **Use existing pipeline-agnostic apps**
  - Apps that program the network using FlowObjectives

- **Write new pipeline-aware apps**
  - Apps can use same string names of tables, headers, and actions as in the P4 program
Performance considerations
Performance overhead

![Diagram showing the process of writing and reading data between apps and switches, involving FlowRule, PiTableEntry, and P4Runtime.TableEntry classes, and the role of PipelineInterpreter and P4info in translation and encoding/decoding.]
Relevant use cases and metrics

• **ONOS in production at major US carrier**
  • Leaf-spine fabric for subscriber access
  • Proactive flow programming, routing-heavy (100s of 1000s of routes)

• **Known pain points**
  • Initial switch provisioning / reboot / failure re-routing
    • Insert/update routes
  • Flow rule reconciliation (periodic, every 10-100 seconds)
    • Read device state and compare with ONOS store

• **Relevant metrics**
  • Read/write operations throughput
  • Latency for batched write operations
  • Memory utilization
Proposed testing methodology

- **Measure relevant metrics with up to 1M routes**
- **Remove overhead of switch implementation (e.g. slow ASIC writes)**
  - Use Stratum “Dummy” switch (yet to be open sourced)
    - Write/read from memory instead of ASIC
  - Or, implement dummy P4Runtime server
    - E.g. in golang or C, easy with gRPC auto-generated stubs
- **Remove/minimize RTT between switch and ONOS**
  - Ideally run ONOS and P4Runtime process on the same machine
Conclusions

• ONOS extended to add support for P4 & P4Runtime
• Maintain backward compatibility with existing northbound APIs via internal translation
  • FlowRule, FlowObjective API → P4Runtime protobuf messages
• Translation adds overhead
• Proposed activity: measure ONOS performance when using P4Runtime in relevant use causes (routing heavy)
Get started: ONOS+P4 tutorial

● Basic
  ○ Learn the basics of P4Runtime and ONOS with hands-on exercises
  ○ https://wiki.onosproject.org/x/MwP2

● Advanced
  ○ Build a leaf-spine fabric based on SRv6 with P4 and ONOS
  ○ https://wiki.onosproject.org/x/xIFfAg
Thanks!