Launching the P4 API Working Group

Antonin Bas (antonin@barefootnetworks.com)
Lorenzo Vicisano (vicisano@google.com)

Motivation for standardizing P4 APIs

- P4 language specifies the dataplane of a wide range of networking devices
- Elements of a P4 pipeline (e.g. tables) need to be managed at runtime to configure the desired forwarding behavior, so runtime APIs need to be designed
- Standard APIs enable silicon independence by providing a common way of controlling all P4-programmable switches

```
action set_nhop(bit<32> nhop) {
    …
}
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr : lpm;
        hdr.meta.vrf_id : exact;
    }
    actions = {
        drop;
        set_nhop;
    }
}
```


```
ipv4.dstAddr=10.0.0.0/8, meta.vrf_id=0xaa ->
    set_nhop(10.0.0.1)
ipv4.dstAddr=10.1.0.0/16, meta.vrf_id=0xaa ->
    set_nhop(10.1.0.1)
ipv4.dstAddr=12.0.0.0/8, meta.vrf_id=0x00 ->
    drop()
```

Scope of the standard runtime APIs

- Provide standard means for:
  - **local control** (NOS integration) => silicon independence
  - **remote control** (SDN controller integration) => switch vendor independence
- Runtime management of P4 tables
- Runtime management of Portable Standard Architecture (PSA) externs (e.g. Counter, Meter, ActionProfile, …)
- Ability to extend the API to support vendor-specific externs
- Minimal session management (remote API only)

Switch configuration?

- For a good end-to-end solution, runtime management of a P4 pipeline is not enough
- We believe that switch configuration (port management, traffic manager configuration, ...) should be in scope
- A good way to achieve this would be to define OpenConfig (http://openconfig.net/) data models in YANG format, specific to switches exposing a P4 runtime API
- Avoid re-inventing the wheel: re-use existing YANG models as much as possible and extend them if necessary

Initial work on one possible runtime API: P4Runtime

- A protocol / program independent API to
  - facilitate vendor adoption
  - enable field-reconfigurability (**ability to push a new P4 dataplane to the device, without any code compilation needed on the switch**)
- Device does not need to be fully P4 programmable
  - **can be used on legacy fixed-function devices**, providing their forwarding behavior can be expressed with a P4 program (for incremental update of hardware)
- Targets a remote controller
  - protobuf + gRPC implementation: well-known, well-supported serialization format + many RPC features for free (e.g. authentication…)

P4Runtime example: adding an entry to a table

```
action set_nhop(bit<32> nhop) {
    …
}
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr : lpm;
        hdr.meta.vrf_id :
exact;
    }
    actions = {
        drop;
        set_nhop;
    }
}
```

```
ipv4.dstAddr=12.0.0.0/8,
meta.vrf_id=0x00
    -> drop()
```

```
table_entry {
  table_id: 33581985
  match {
    field_id: 1
    lpm {
      value: "\f\000\000\000"
      prefix_len: 8
    }
  }
  match {
    field_id: 2
    exact {
      value: "\000\000"
    }
  }
  action {
    action {
      action_id: 16812204
    }
  }
}e
```

P4 table definition

Logical view of entry

Protobuf entry encoding

P4 API WG

- Charter document to be published soon
- p4-api@lists.p4.org mailing list
- P4Runtime is open-source and in active development on Github
  - https://github.com/p4lang/PI
- We welcome new ideas and contributions, send us an email if you want to participate