Google Cloud

P4 Language - Ideas for Future Directions

Based on Use-cases of P4 at Google

Stefan Heule <heule@google.com>
Google, Network Infrastructure

P4 Language Brainstorming
June 22, 2020

Our Motivation to Use P4

- Precisely model the expected switch behavior (fixed-function switch)
  - Get switch simulator for testing
  - Ability to build automated tools that can reason about switch, e.g. to automatically generate interesting packets

- Clear control plane API
  - Ability to test switch through fuzzing
  - Debuggability

Control plane and Data plane are not the same

Dataplane

- “Everything is bits”

Controlplane

- API requires names, not implementation (e.g. match field name vs match expression)
- Bits have meaning, e.g. a MAC address vs an IP
- Some values aren’t bits, like port names

Example

Controller uses strings for ports

```
@p4runtime_translation("...", string)
type bit<10> port_id_t;
```

Translated types can only support exact or optional

Controller needs fixed IDs for API stability.

To define control plane API, we want clear names

```
@id(SOME_TABLE_ID)
table some_table {
  key = {
    metadata.ingress_port : optional @id( 1) @name("ingress_port");
    headers.ipv4.dst_addr : lpm @id( 2) @format(IPV4_ADDRESS)  @name("ipv4_dst");
  }
  actions = {
    drop;
    set_nexthop_id;
    set_wcmp_group_id;
  }
  const default_action = drop;
}
```

Different “bits” have different meaning in the controller. Indicate format (for pretty-printing, debuggability)

What could P4 2.0 look like?

P4 program clearly allows specification of control plane API.

- Naming, formatting of values, ID allocation are not an afterthought but first-class.
- Good support for translated types: program can only do equality-comparisons on them (hence *optional* and *set* match kinds)
- Better support to say something is an IPv4 address once, and then being able to use this in many places (currently we use @format in every table, instead of once in the header). Similar to [p4lang/p4-spec#815]

What could P4 2.0 look like?

Match kinds should be clearly specified (right now spec says almost nothing about their semantics)

- Including ability to have const tables for new match kinds. Unclear how const tables for, say, *set* match kind would look.

Thank you

Stefan Heule <heule@google.com>

Google Cloud