Leveraging P4 for Fixed Function Switches

Konstantin Weitz (konne@google.com)
Waqar Mohsin (wmohsin@google.com)
Amin Vahdat (vahdat@google.com)
Stefan Heule (heule@google.com)
Lorenzo Vicisano (vicisano@google.com)
Google, USA

To use a fully-programmable switch chip, one has to describe the switch pipeline using P4. However, Google, like most of the industry, still relies on fixed-function switch chips in our infrastructure. This will continue for years to come. We believe that using a single *contract* to specify forwarding behavior across our entire infrastructure has huge value, beyond the benefits of dynamic switch reconfiguration in a subset of the fabric. In this talk, we show how modeling a pipeline in P4 can provide value even in deployments that consist of mostly fixed function switches. The advantages are:

1) Having a P4 program that clearly describes the semantics of our switches enables automated validation [1][2]. This approach is becoming ever more important as we move toward a world with more heterogeneous switches, more stringent availability requirements, and faster release cycles. The traditional, manual approach to writing tests cannot scale further.

2) Making our requirements use-case-centric, rather than based on vendor capability, simplifies portability across vendors. Our P4 programs model what we need from the switch in a particular role, not what the switch provides. For example, we have a logical encap table of size 13, because that's all we need; or we have multiple logical tables, which most platforms will implement with a single physical table. This makes it easier to operate a heterogeneous fleet, which could consist of both fixed-function and programmable switches.

3) By using a P4 program to clearly describe our requirements for a switch, we decuple the specification from the implementation, potentially removing “vendor lock in”: The combination of portable specification and standard APIs (P4Runtime) gives us access to a larger pool of vendor switches.

4) Many fixed function switch chips have some programmable parts. We have a compiler that inspects our P4 programs, and generates the appropriate switch configuration for these programmable subsets. Our compiler also verifies that our logical tables fit within the limits of the hardware.

5) In some cases, we need our switches to provide as many table resources as possible. In such cases, our P4 program is instantiated with the limit for each platform. Our SDN controller can then inspect these limits, for example, to make tradeoffs between optimal link utilization and table resource consumption.

In conclusion, we have found that describing our switch pipelines in P4 enables us to perform automated testing, operate a heterogeneous fleet, make our switches a commodity, generate configuration, and improve our hardware utilization.

[1]: Nötzli et al. p4pktgen: Automated Test Case Generation for P4 Programs. SOSR '18.
[2]: Freire et al. Uncovering Bugs in P4 Programs with Assertion-based Verification. SOSR '18.