Leveraging P4 for Fixed Function Switches

Konstantin Weitz
konne@google.com

Stefan Heule
heule@google.com

Waqar Mohsin
wmohsin@google.com
P4 on Programmable Switches

P4 program determines what the Hardware does
P4 on Fixed-Function Switches

Hardware determines what the P4 program does

Fixed Parser → Virtual Routing and Forwarding → L3 Admit → L2 Routing → Access Control Lists → Fixed Deparser

L3 Admit

L3 Routing

Access Control Lists
P4 on Fixed-Function Switches

Hardware determines what the P4 program does

But, only model what we need:
- skip unused features (e.g. L2)
- tables only include actually used keys and actions
- table sizes are what we use
- for configurable aspects, only model our configuration

...
Why would you want to do this?

Clear *contract* of switch behavior:
- Enables operation of a heterogeneous fleet
- Automatically generate switch config
- Enables automated switch validation
Why would you want to do this?

Clear *contract* of switch behavior:

- Enables operation of a heterogeneous fleet
- Automatically generate switch config
- Enables automated switch validation
Automated Switch Validation
Automated Switch Validation

Test inputs are automatically generated, either from production data, or by analyzing our P4 programs.
Automated **Switch** Validation

We validate a single switch chip, not the whole network.
Automated Switch Validation

Test outputs are compared to a P4 program simulation.
How do we test the switch?

**Replay** production flows/groups

**Fuzzer** to randomly create flow/group insert/delete requests

**ATPG**: Automated Test Packet Generation
Controlplane Fuzz Testing
Controlplane Fuzzing

Randomly generate flow requests according to P4 program grammar
- Mostly generate well-formed requests
- Sometimes generate ill-formed ones
- Intuition: Need to be well-formed enough to not get rejected early

Send flow to switch, check that they are handled correctly
- E.g. well-formed insert must succeed (unless resource exhausted or already present)
- P4 allows us to accurately predict the expected error (or success)
Automated Test Packet Generation
Automated Test Packet Generation

Packet Generator → Input Packets → Flows → P4 Simulator (BMv2) → Switch → Actual Output Packets

Legend:
- Software
- Controlplane
- Dataplane

Expected Output Packets → Verify Match

P4Runtime

Google
## Generation Strategy: Hitting every flow on the switch

### VRF Classifier

<table>
<thead>
<tr>
<th>EthType</th>
<th>SrcMac</th>
<th>Port</th>
<th>Set VRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x800</td>
<td>aa:bb:cc:dd:ee:ff</td>
<td>*</td>
<td>1337</td>
</tr>
<tr>
<td>0x800</td>
<td>*</td>
<td>4</td>
<td>42</td>
</tr>
</tbody>
</table>

### IPv4 LPM

<table>
<thead>
<tr>
<th>VRF</th>
<th>DstIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>10.152/24</td>
</tr>
<tr>
<td>42</td>
<td>10.152/16</td>
</tr>
</tbody>
</table>

VRF == 42 & DstIP[32:16] == "10.152"

& !(VRF == 42 & DstIP[32:8] == "10.152.8") & !(...) // avoid all other IPv4 LPM flows

Want to hit this flow

// encode VRF assignment

& ((!(EthType == 0x800 & SrcMac == "aa:bb:cc:dd:ee:ff")
  & (EthType == 0x800 & Port == 4)) → VRF == 42)

// hit target IPv4 LPM flow

SAT solver finds packets to satisfy the formula
Dataplane Testing: why SAT works

- Everything is finite
  (no lists, loops, recursion, etc)

- Switch semantics are rigorously defined in the P4 program
Dataplane Testing: why it works

Test oracle: Clear semantics allow simulator to precisely predict switch behavior

Test generation: Semantics are simple enough that tools can reason about them automatically

Lack of formal and computer-readable specification makes both difficult to do automatically
What kind of Bugs did we find?

- Bugs in the Switch
- Bugs in our SDN Controller
- Bugs in our P4 specs
- Bugs in BMv2
Conclusion
Key Takeaways

P4 provides a clear contract of switch behavior:
- Enables operation of a heterogeneous fleet
- Can be used to generate switch config
- Enables automated switch validation
  (it's fast and finds a broad spectrum of bugs)

We're hiring!
Email: {konne, heule, wmohsin}@google.com