Automated Test Case Generation from P4 Programs

Chris Sommers (presenter)
Kinshuk Mandal
Rudrarup Naskar
Prasenjit Adhikary

June 2018
The Need: Test any arbitrary protocol, conveniently, at line rates

Programmable Data plane (exemplified by P4)

P4 Enables:
- Protocol Independence
- Handle existing and future protocols
- Target Independence
- Line-rate processing

THE NEXT STEP IN SDN

PROBLEM: How do you test a new protocol with existing line-rate testers?
- new protocols not standardized yet, are experimental or proprietary
- tools don’t generally anticipate unknown protocols, or else handle them inadequately

SOLUTION:
The P4 code which defines the function of a device, can also act as the specification for the test-tool.

Thus we can achieve a protocol-independent “protocol test tool”. *

* Keysight has HW based tools for data plane testing
WORKFLOW
Overview of IxNetwork
IxNetwork + Physical Test Chassis

Test Console /IxNetwork Client

Test Control

Ixia Chassis

Packet Engine

Load Modules Up to 400GbE

Traffic

Device Under Test (DUT)
IxNetwork + Virtual Test Chassis

Virtual Chassis

Ixia vChassis
eth1

Ixia vModule
eth0 eth1 eth2

Ixia vModule
eth0 eth1 eth2

Virtual Packet Engine

VNF* VM Under Test
eth0 eth1 eth2

Virtual DUT, e.g. BMv2

Physical DUT

Control vSwitch

Test vSwitch

HOST

NIC1

Lab Mgmt Network

IxNetwork Console

Physical Device Under Test

Physical

Virtual

Virtual Network Function

Physical

Physical

Virtual

Virtual

Virtual
What did we do?
Enhancing IxNetwork to be P4-Aware

- Create a new backend for the p4c compiler: p4c-ixia. Output is test-case metadata

- Enhance IxNetwork to embed and launch p4c-ixia and to import the test-case metadata

- Enhance IxNetwork to translate test-case metadata into test data streams utilizing our packet engines

- Existing load modules (physical and virtual) are already highly programmable and largely protocol-agnostic. *No modifications were required on the packet engines.*

- This also allows both our physical and virtual packet testers to support p4 testing.
Metadata Packet Structures & Field Values:

- All possible valid sequence of protocol headers (packet templates).
- "Header Stacks" information (eg: multiple MPLS labels)
- Erroneous Packets (eg: hitting the reject rules, exceeding the boundary conditions)
- Packet fields values to test the limit of "Verify" condition
- Packet structure and field values needed to execute "Key Set" for "Select"
- Dealing with constant entries in "P4 Table".
Calculator Protocol – An arbitrary data-plane as test case

The Data Plane

- Header stack
- Validation: pkts with 1~3 labels

- OPERATOR is an operation to perform. It is of 8 bits.
  - '+': (0x2b) Result = Left Operand + Right Operand
  - '-': (0x2d) Result = Left Operand - Right Operand
  - '&': (0x26) Result = Left Operand & Right Operand
  - '|': (0x7c) Result = Left Operand | Right Operand
  - '^': (0x5e) Result = Left Operand ^ Right Operand

https://github.com/p4lang/tutorials/blob/master/P4D2_2017_Spring/exercises/calc/solution/calc.p4
Video Demonstration
## Results: Templates (Positives & Negatives)

<table>
<thead>
<tr>
<th>Stack name</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;(001)-ETHERNET.MPLS.MPLS.MPLS.MPLS - REJECT&quot;</td>
<td>Rejected as 4th MPLS stack not supported.</td>
</tr>
<tr>
<td>&quot;(002)-ETHERNET.MPLS.MPLS.MPLS.ETHERNET.P4CALC&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(003)-ETHERNET.MPLS.MPLS.MPLS.ETHERNET.P4CALC - REJECT&quot;</td>
<td>Rejected by p4calc version (0x503402 accepted type)</td>
</tr>
<tr>
<td>&quot;(004)-ETHERNET.MPLS.MPLS.MPLS.ETHERNET - REJECT&quot;</td>
<td>Rejected by inner eitherType (0x1234 accepted type – calc protocol)</td>
</tr>
<tr>
<td>&quot;(005)-ETHERNET.MPLS.MPLS.ETHERNET.P4CALC&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(006)-ETHERNET.MPLS.MPLS.ETHERNET.P4CALC - REJECT&quot;</td>
<td>Rejected by p4calc version (0x503402 accepted type)</td>
</tr>
<tr>
<td>&quot;(007)-ETHERNET.MPLS.MPLS.ETHERNET - REJECT&quot;</td>
<td>Rejected by inner eitherType (0x1234 accepted type – calc protocol)</td>
</tr>
<tr>
<td>&quot;(008)-ETHERNET.MPLS.ETHERNET.P4CALC&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(009)-ETHERNET.MPLS.ETHERNET.P4CALC - REJECT&quot;</td>
<td>Rejected by p4calc version (0x503402 accepted type)</td>
</tr>
<tr>
<td>&quot;(010)-ETHERNET.MPLS.ETHERNET - REJECT&quot;</td>
<td>Rejected by inner eitherType (0x1234 accepted type – calc protocol)</td>
</tr>
<tr>
<td>&quot;(011)-ETHERNET - REJECT&quot;</td>
<td>Rejected by outer eitherType (0x8847 accepted type)</td>
</tr>
</tbody>
</table>
Results: What does the Ixia Traffic Engine see & do?

- User friendly mechanism to vary any protocol fields.
- Use Ixia’s powerful pattern editor to vary the fields. Underneath Ixia UDF(s) are used to support variation.
- Flexibility of any fields to track (including the new protocol) at Line rate.
- Ingress and Egress tracking support.
- Track on meta data (Frame size, Flow Group etc.).
- Facility to utilize the latency bin(s).
- Flow Grouping - lowest level of control on Frame rate / size / start & stop

In this way we achieve a protocol independent “protocol test tool”
Invalid field values (for negative test case 003)

- Fields are pre-populated with invalid values.
- For Calculator Protocol, type should be ASCII 'P' (decimal 80) and Subtype should be ASCII '4' (decimal 52).
- Fields are pre-populated with all 8-bit values except the valid ones.
- Pattern Editor customization
- Similarly, for version field, field is pre-populated with all values other than 2, to verify target behavior for reject scenario.
- And so forth…
Results: Tracking based on arbitrary fields

- Packet generation at Line rate.

- Drill down statistics based on the tracking fields (including arbitrary fields in arbitrary protocol)

- Simulates thousands of packets in specific order and verify correct order and latency
What's Next?

- Focus on testing, stability and react to community feedback.
- Automatic packet decoder

Frame 23: 200 bytes on wire (1600 bits), 200 bytes captured (1600 bits) on interface 0
- Ethernet II, Src: aa:00:00:00:00:01 (aa:00:00:00:00:01), Dst: 00:0c:29:09:f0:08 (00:0c:29:09:f0:08)
- MultiProtocol Label Switching Header, Label: 10016, Exp: 0, S: 0, TTL: 64
- MultiProtocol Label Switching Header, Label: 20016, Exp: 0, S: 0, TTL: 64
- MultiProtocol Label Switching Header, Label: 30016, Exp: 0, S: 1, TTL: 64
- Ethernet II, Src: aa:00:00:00:00:01 (aa:00:00:00:00:01), Dst: 00:0c:29:09:f0:08 (00:0c:29:09:f0:08)
- Calculator Protocol
  - version: 2
  - operator: PLUS (0x2b)
  - Left Operand: 100
  - Right Operand: 10
  - Result: 110

- Stateful Fuzzing of arbitrary protocol
Future possibility - Control Plane Integration
Thank You

Questions?