Toward Consistent State Management of Adaptive Programmable Networks Based on P4

SIGCOMM – NEAT Workshop 2019,
Beijing, August 19th, 2019

Mu He
Mu.he@tum.de

Andreas Blenk, Stefan Schmid, Wolfgang Kellerer

This work is part of a project that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program grant agreement No 647158 – FlexNets (2015 – 2020).
New Applications and Technologies

Augmented Reality

Virtual Reality

Tactile

Vehicular Networks

Requirements on reliability, throughput, latency, programmability, and customizability!

Programmable Networks with P4

- Program your own data plane [1]
  - Protocol independent
  - Target independent
  - Field reconfigurable

- Advantages in many use cases
  - In-band network telemetry [2]
  - Flow monitoring [3]
  - Load balancing [4]
  - High throughput data center [5]

Reconfiguration is needed!

Field reconfigurability + Data plane states

How to manage them?
State Consistency during Network Reconfiguration

- Stateful firewall based on packet counts (More than 100)
- State: # packets per flow

<table>
<thead>
<tr>
<th>Flow</th>
<th># PKT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Incorrect copy of # PKT before reconfiguration:

<table>
<thead>
<tr>
<th>Flow</th>
<th># PKT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Correct copy of # PKT after reconfiguration:

<table>
<thead>
<tr>
<th>Flow</th>
<th># PKT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
New Requirements

**Completeness**

- All necessary state variables
- Updated version

**Rapidness**

- Finish as fast as possible
- Agility of data plane
Our Proposal: P4State Analyzer

- States identification
  - Collect all state usage
  - Bind states to tables and if-conditionals

- Control Flow Graph (CFG) construction
  - Abstract match + action pipeline

- Control Flow Graph pruning
  - Delete stateless nodes

- Path and role identification
State Variables in P4: Taxonomy

- Various types of states in P4 context
  - Table entries ↔ control plane
  - Temporary metadata: local information while processing each packet
  - **Stateful variables**: register

- Persistent beyond the packet processing loop, e.g.,
  - Packet counter
  - Port status
  - Pipeline control flag
State Variables in P4: Taxonomy

- Register = An array with values
  - Register type
  - Register entry

- Flow-based registers
  - Information per-flow (different granularity)
  - Large number of entries
  - Migrated through data plane or control plane

- Device-based registers
  - Device and pipeline information
  - Migrated through control plane

```c
register <bit <8>>(1024) R1;
```

R1 with 1024 entries, each of 8 bits
States Identification

- Input: P4 program
- Output: Set of tables and if-conditionals with register access

- Compile the program to a JSON file
  - Consists definitions of all pipeline components
- Parse JSON file
  - Collect all register definitions
  - Collect all header, action, table, if-conditional definitions
  - Associate registers to tables and if-conditionals
Control Flow Graph (CFG)

- Describes the packet processing pipeline
- Composed of **tables** and **if-conditionals**
- "Start" and "Stop" node

```plaintext
| table ipv4_lpm {
|    key = {
|        hdr.ipv4.dstAddr: lpm;
|    }
|    actions = {
|        ipv4_forward;
|        drop;
|    }
|    size = 1024;
|}
| apply {
|    if (hdr.ipv4.isValid()) {
|        ipv4_lpm.apply();
|    }
|}
```
Stateful CFG Construction

- From “Start”, recursively explore the child nodes of each node, until “Stop” is reached → Find a path
- Merge the common parts of all paths
CFG Pruning

- Why pruning?
  - Original CFG is hard to digest
  - Stateless nodes in CFG have no effect

- Inspired by Thin Slicing [1]

- Two steps
  - 1. Remove all nodes without state access
  - 2. Merge consecutive tables with same state access on a single path

cfg pruning

- why pruning?
  - original cfg is hard to digest
  - stateless nodes in cfg have no effect

- inspired by thin slicing [1]

- two steps
  1. remove all nodes without state access
  2. merge consecutive tables with same state access on a single path

Path & Role Identification

- Why identification?
  - Single P4 program consists of multiple functions
  - Different functions enabled on different P4 switches

- Input
  - Pruned CFG
  - Table entries during data plane initialization
  - Domain knowledge
  - (Human intervention)

- Output
  - States that need to be migrated for a certain P4 switch
  - (Migration schedule)
Evaluation on HULA

- Traffic engineering in data center networks [1]
- Probing
  - Maintain the current best paths between TOR switches considering queue length
- Forwarding
  - Send traffic on the best paths

- Probing: R3, R4
- Forwarding: R1, R2
- TOR switches: Probing + Forwarding
- Core switches: Forwarding

General Performance: Algorithm Runtime

- Comparable CFG construction time and CFG pruning time
- Synthetic: scales with # tables, less than 100 ms
- Real: depends on pipeline structure (LinearRoad has higher LoC, but Dapper has more branches)
P4State analyzer can **quickly** and **successfully** recognize the register types that need migration during data plane reconfiguration

- Avoid unnecessary register migration
- Output CFG with states that allows human intervention

**Next steps**

- Only migrate valid register entries
- Scheduling of register migrations

<table>
<thead>
<tr>
<th>Program</th>
<th>LoC</th>
<th>Types of Reg.</th>
<th># Reg. Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flowlet</td>
<td>203</td>
<td>2</td>
<td>16384</td>
</tr>
<tr>
<td>netpaxos</td>
<td>210</td>
<td>6</td>
<td>256002</td>
</tr>
</tbody>
</table>
Thank you! & Questions?