Programming Language for Switches

ECE/CS598HPN

Radhika Mittal
Conventional SDN

• Very flexible control plane in software.

• Interacts with dataplane through OpenFlow.

• Dataplane flexibility limited by:
  • what OpenFlow supports.
  • what the underlying hardware can support.
# OpenFlow Support

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th># Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>OF 1.0</td>
<td>Dec 2009</td>
<td>12</td>
</tr>
<tr>
<td>OF 1.1</td>
<td>Feb 2011</td>
<td>15</td>
</tr>
<tr>
<td>OF 1.2</td>
<td>Dec 2011</td>
<td>36</td>
</tr>
<tr>
<td>OF 1.3</td>
<td>Jun 2012</td>
<td>40</td>
</tr>
<tr>
<td>OF 1.4</td>
<td>Oct 2013</td>
<td>41</td>
</tr>
</tbody>
</table>
Programmable Switches

PISA: Protocol Independent Switch Architecture

- RMT:
  - Programmable parsers.
  - Reconfigurable match-action tables.

- Intel FlexPipe

- Cavium Xpliant
What was missing?

An interface to program such switches.
P4 Goals

• Protocol independence
  • Switches are not tied to specific packet formats.

• Reconfigurability
  • Controller can redefine packet parsing and processing in the field.

• Target Independence
  • User program need not be tied to a specific hardware.
  • Compiler’s job to do the mapping.
P4 vs OpenFlow
Components of a P4 program

- Header definitions
- Parser definition
- Tables: what fields to match on, and which action to execute/
- Action definition.
Example
Example

header ethernet {
    fields {
        dst_addr : 48; // width in bits
        src_addr : 48;
        ethertype : 16;
    }
}

header mTag {
    fields {
        up1 : 8;
        up2 : 8;
        down1 : 8;
        down2 : 8;
        ethertype : 16;
    }
}

header vlan {
    fields {
        pcp : 3;
        cfi : 1;
        vid : 12;
        ethertype : 16;
    }
}

```c
control main() {
  // Verify mTag state and port are consistent
  table(source_check);

  // If no error from source_check, continue
  if (!defined(metadata.ingress_error)) {
    // Attempt to switch to end hosts
    table(local_switching);

    if (!defined(metadata.egress_spec)) {
      // Not a known local host; try mtagging
      table(mTag_table);
    }

    // Check for unknown egress state or
    // bad retagging with mTag.
    table(egress_check);
  }
}
```
table mTag_table {
   reads {
      ethernet.dst_addr : exact;
      vlan.vid : exact;
   }
   actions {
      // At runtime, entries are programmed with params
      // for the mTag action. See below.
      add_mTag;
   }
   max_size : 20000;
}
Example

```c
action add_mTag(up1, up2, down1, down2, egr_spec) {
    add_header(mTag);
    // Copy VLAN ethertype to mTag
    copy_field(mTag.ethertype, vlan.ethertype);
    // Set VLAN’s ethertype to signal mTag
    set_field(vlan.ethertype, 0xaaaa);
    set_field(mTag.up1, up1);
    set_field(mTag.up2, up2);
    set_field(mTag.down1, down1);
    set_field(mTag.down2, down2);

    // Set the destination egress port as well
    set_field(metadata.egress_spec, egr_spec);
}
```
Example

• This was the edge switch’s mTag match-action table.

• What will the core do?
  • Table will have ternary match on mTag
  • Action will be mTag_forward
    • Forward on specified port.
    • The rule about which mTag matches to which port is part of the configuration file.
P4 Compiler

• If the target is a fixed-function switch?
  • Check if specified parser and match-action tables are supported.
  • If not, return error.

• If target is a software switch?
  • Full flexibility to execute specified program.
  • May use specific software data structures for optimizations.

• If target is an RMT switch?
  • Figure out table layout
    • mapping logical stages to physical ones.
    • When to use RAM vs TCAM
  • If tables don’t fit, an action not support, etc: return an error.
Your Opinions

• Pros
  • Identifies primitives for dataplane programmability.
  • Much needed interface (for programmable switches).
  • Sweet-spot between flexibility and performance
    • More future-proof than OpenFlow
    • More constrained than Click
  • Useful features:
    • Target-independence
    • Maintain state via metadata.
  • Example shows ease of use.
Your Opinions

• Ideas
  • Usecases
    • Monitoring
    • Load balancing
  • Compare OpenFlow and P4 for different usecases
  • How to optimize P4 code compilation?
  • A debugging tool for P4
  • Explore the limitations of P4
Your Opinions

- **Cons**
  - What happens during reconfiguration?
  - Performance penalty of expressiveness?
    - No evaluation benchmark
  - Why the imposed limitations?
  - Is it really target independent?
  - What is the minimum required hardware support?
  - What are the limitations of P4?
Is P4 Turing-complete?
Limitations of P4 and PISA model
Event-Driven Packet Processing

Stephen Ibanez, Gianni Antichi, Gordon Brebner, Nick McKeown

HotNets 2019
Baseline PISA

Restricted to packet ingress and egress events.
Limitations

- Periodic events
  - Generate probe packets.
  - Reset counters.

- Other non-packet events
  - Link failure.
Trigger on events, not packets

- Packets generate events when traversing the pipeline:
  - Ingress, enqueue, dequeue, egress, overflow, etc.

- Enable time-based events:
  - Periodic timers.

- Enable other events:
  - Link status change.
Updated Switch Architecture
Challenges

• More event threads, more state coordination.

• Locally record state updates.

• Aggregate when memory bandwidth is available.
Generic External Memory for Switch Dataplanes

Daehyeok Kim, Yibo Zhu, Changhoon Kim, Jeoungkeun Lee, Srinivasan Seshan

HotNets 2018
Basic Idea

• Switches require high memory bandwidth.
  • Use fast, but expensive on-chip SRAM and TCAM.
  • Limited in size.

• Memory size could be a limiting factor for many applications.

*Let’s access endhost memory remotely*....
Queuing is not yet fully programmable.