From P4 to OpenFlow

Ben Mack-Crane, Peter Musgrave

Corsa Technology

OF-PI: A Protocol Independent Layer

- P4 can be used to define a datapath configuration.
- Need a run-time protocol for managing datapath behaviors (e.g., flows).
- OpenFlow is an obvious candidate.
- How do we get from P4 to OpenFlow?

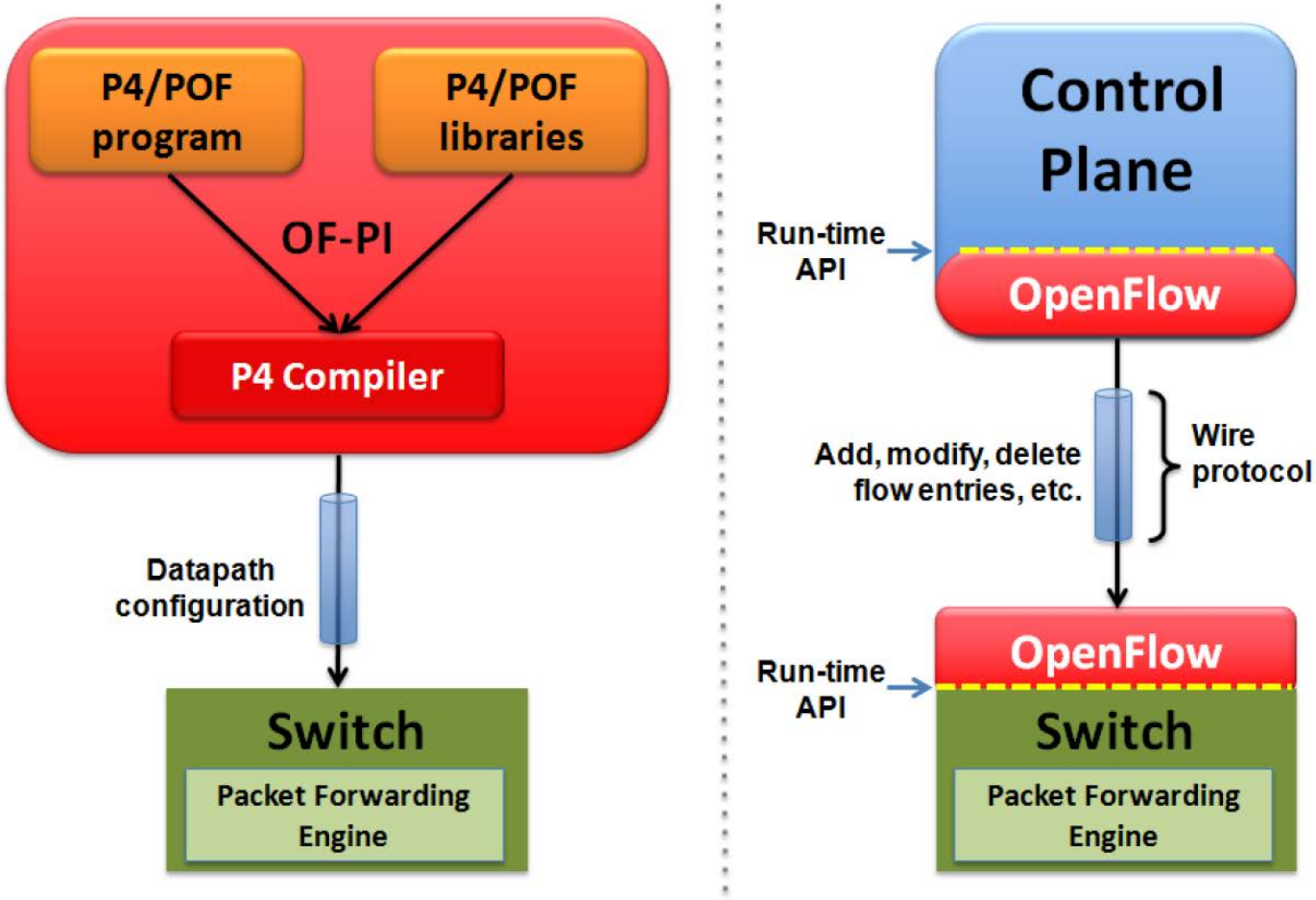
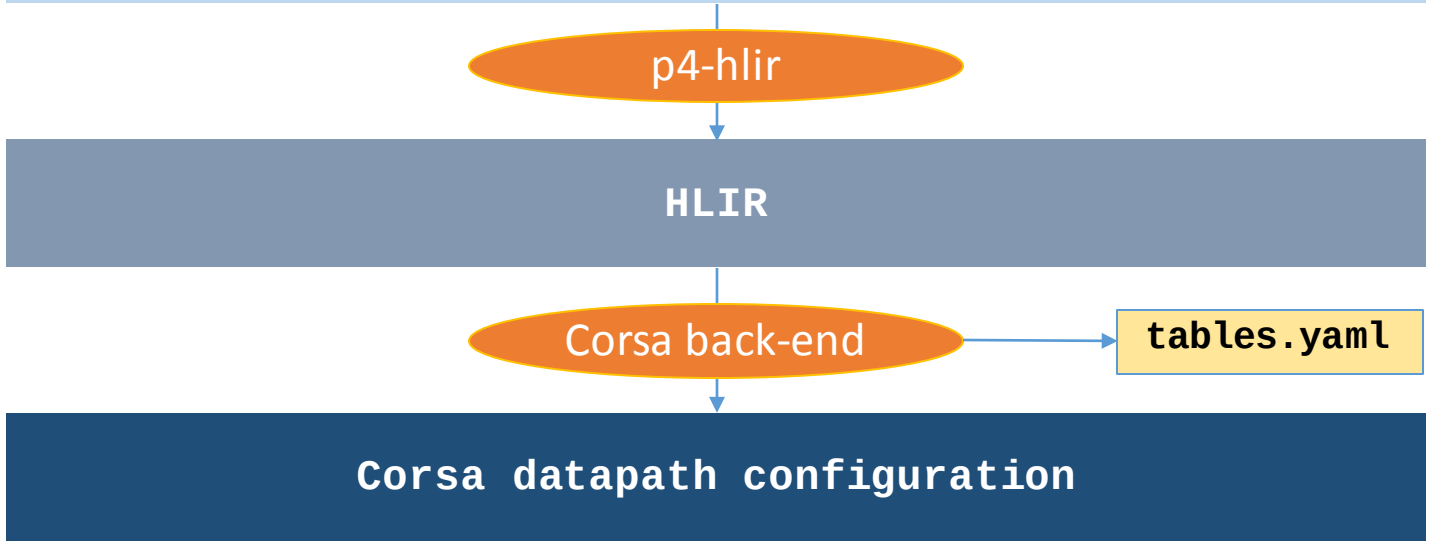


Figure 4: OF-PI has two stages, Configuration and Run-time.

P4 Compilation

- Using the P4 front-end compiler (p4-hlir) we generate HLIR for the datapath configuration.
- The Corsa back-end compiler generates a configuration file for the Corsa dataplane.
- The Corsa back-end compiler also generates a YAML file containing table descriptions for P4 tables.

```
table l3_dscp {
  reads {
    ipv4.diffserv: exact;
    ipv4.dstAddr: exact;
    vlan.vid: exact;
  }
  actions {
    miss_l3_dscp;
    set_vid_queue_meter_port;
  }
}
…
action set_vid_queue_meter_port(vlan_vid, queue_id, meter_id, port_out)
{
   modify_field( vlan.vid, vlan_vid);
   modify_field( corsa_metadata.queue_id, queue_id);
   modify_field( corsa_metadata.egress_port, port_out);
   meter(color_meter, meter_id, corsa_metadata.meter_color);
   add_to_field( ipv4.diffserv, corsa_metadata.meter_color);
}
```



Mapping to OpenFlow

- The tables.yaml file is used to create OpenFlow run-time control code.
  - Map the fields to OpenFlow Match Fields
  - Flatten P4 actions and map P4 primitives to OpenFlow actions
  - Not specific to the Corsa dataplane

```
l3_dscp :
   id : 1
   match :
       fields:
       - 'ipv4.diffserv' :
         type  : P4_MATCH_EXACT
         width : 8
       - 'ipv4.dstAddr' :
         type  : P4_MATCH_EXACT
         width : 32
       - 'vlan.vid' :
         type  : P4_MATCH_EXACT
         width : 12
   actions :
     - miss_l3_dscp :
       - goto_table : 2
     - set_vid_queue_meter_port :
       fields:
         - vlan_vid : 12
         - queue_id : 2
         - meter_id : 10
         - port_out : 6
       - set_field :
         - vlan.vid : vlan_vid
       - set_queue : queue_id
       - output : port_out
       - meter : meter_id
       - add_field :
         - ipv4.diffserv : 1
```

OFPXMT_OFB_IP_DSCP

OFPXMT_OFB_IPV4_DST

OFPXMT_OFB_VLAN_VID

OFPAT_SET_FIELD

OFPAT_SET_QUEUE

OFPAT_OUTPUT

OFPAT_METER

OFPAT_???

Some Challenges

P4 does not have have a group table concept	Similar to an index table but can recurse
P4 allows one action for a match while OpenFlow allows action lists	Flatten P4 actions and map primitives to OpenFlow actions to create action list
P4 does not support in-context control (i.e., GOTO_TABLE)	Control flow analysis generates OpenFlow GOTO_TABLE instructions where needed
P4 allows immutable control logic	Map to (immutable) OpenFlow tables (e.g., standard ETH_TYPE match and next table selection)

Some Challenges

OpenFlow does not support expressions in action parameter lists	Extend OpenFlow
OpenFlow does not support range match	Extend OpenFlow
How to reference new headers in OpenFlow?	OpenFlow allows Experimenter ID codes for Match Fields and Actions (can we do better?)
OpenFlow OUTPUT in an action list (combination of clone and set egress port)	Change OpenFlow to use EGRESS_PORT metadata and more explicit clone action?
OpenFlow allows multiple match types in a single table	Can we map multiple P4 tables into one OpenFlow table?

Another way to look at the problem

- In this exercise we are mapping from an implementation (in P4) to a run-time control interface definition (like an OpenFlow TTP).
  - This assumes the implementation satisfies the run-time control requirements
  - Where do these run-time control requirements come from? Network design!
  - What is the network design and where is it captured?

- Alternatively we might capture the run-time control requirements as an OpenFlow TTP and use that to generate a P4 template.
  - I.e., “From OpenFlow to P4”

- A Suggestion –

  Develop P4 and OpenFlow in concert to provide compatible capabilities for SDN datapath programming and run-time control