PN Lab 2

Due: 11/24

Outline

- Introduction
- Supplement

Lab2

- spec: https://hackmd.io/@83VCk1VZSiCVJJdX97QEMg/rkjSehQNv
- recommended tutorial: https://hackmd.io/@83VCk1VZSiCVJJdX97QEMg/ryDZ_xKdP
- vs lab1
  - Platform: OvS → bmv2
  - Protocol: openflow → P4Runtime

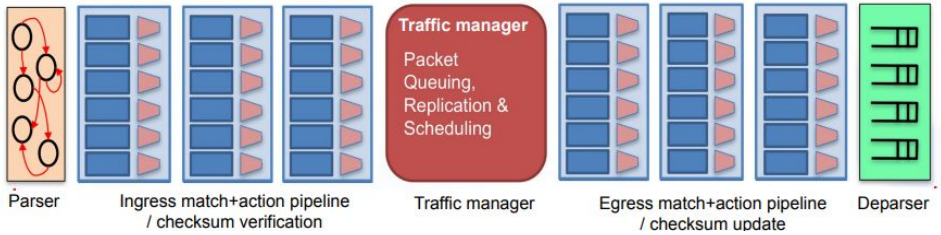
API	Target-independent Same API works with different switches/vendors	Protocol-independent Same API allows control of new protocols	Pipeline-independent Same API allows control of many pipelines formally specified
OpenFlow	✔	✘ Protocol headers and actions hard-coded in the spec	✘ Pipeline specification is not mandated (TTPs did not solve the problem)
Switch Abstraction Interface (SAI)	✔	✘ Designed for legacy forwarding pipelines (L2/L3/ACL)	✘ Implicit fixed-function pipeline
P4Runtime	✔	✔	✔ (with P4)

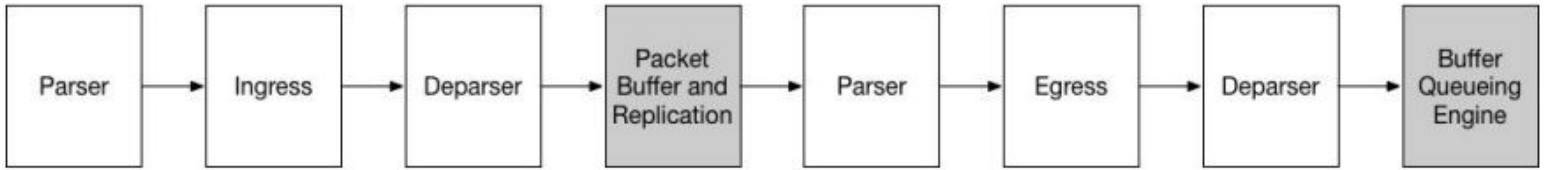
Supplement

- bmv2
- P4Runtime
- ONOS & P4 workflow

bmv2 (behavioral model version 2)

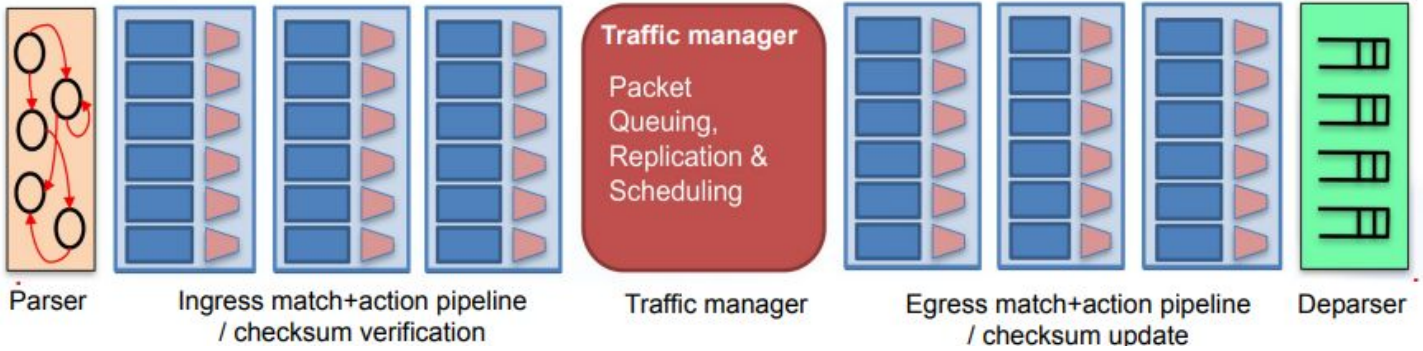
- the second version of the reference P4 software switch, nicknamed bmv2
- It is meant to be used as a tool for developing, testing and debugging P4 data planes and control plane software written for them.
- several variations architecture of bmv2 framework
  - simple_switch, simple_switch_grpc
  - psa_switch





simple_switch

- can execute most P4_14 and P4_16 programs
- The include file v1model.p4 defining the v1model architecture for P4_16 programs
  - The v1model architecture was designed to be identical to the P4_14 switch architecture, enabling straightforward auto-translation from P4_14 programs to P4_16 programs that use the v1model architecture.



P4 program template

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
          out headers hdr,
          inout metadata meta,
          inout standard_metadata_t smeta) {
  ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
   ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  ...
}
```

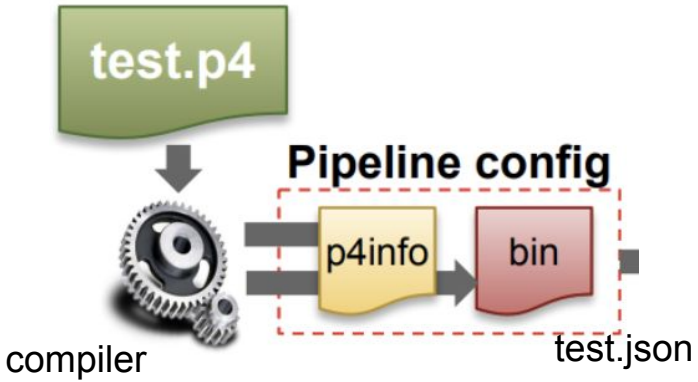
```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
  ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```

Running your P4 program in bmv2 (simple_switch)

1. Use **p4c** to compile the P4 code into a json representation which can be consumed by the software switch
   a. ex: p4c --target bmv2 --arch v1model --std p4-16 <prog>.p4
   b. this example only outputs .json
2. Run simple switch
   a. sudo ./simple_switch -i 0@<iface0> -i 1@<iface1> <prog>.json
3. Configure your switch through switch CLI, the local control plane
   a. ./runtime_CLI.py --thrift-port 9090
      i. The CLI connect to the Thrift RPC server running in each switch process
      ii. 9090 is the default value

p4c

- a reference compiler for the P4 programming language.
- It supports both P4-14 and P4-16
- If your program successfully compiles, the command will create files with the same base name as the P4 program you supplied
  - a file with suffix .p4i, which is the output from running the preprocessor on your P4 program.
  - a file with suffix .json that is the JSON file format expected by BMv2 behavioral model simple_switch.

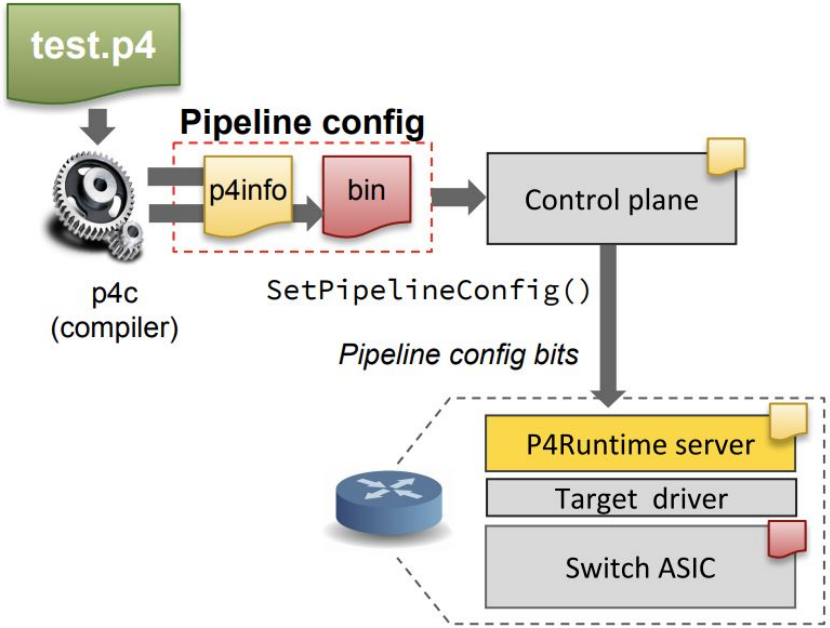


simple_switch_grpc

- vs simple_switch
  - can also accept TCP connections from a controller, where the format of control messages on this connection are defined by the **P4Runtime API** specification.
  - can still also accept connections from controllers using the **Thrift API**, but this is primarily intended for debug purposes
- It is recommended that you do not attempt to have multiple controllers controlling the same target device, one controller using the Thrift API, and another using the P4Runtime API.
  - Warning: because the capabilities of the Thrift server overlap with those of the gRPC/P4Runtime one (e.g. a table management API is exposed by both), there could be inconsistency issues when using both servers to write state to the switch. For example, if one tries to insert a table entry using Thrift, the same cannot be read using P4Runtime. In general, to avoid such issues, we suggest to use the Thrift server only to read state, or to write state that is not managed by P4Runtime.

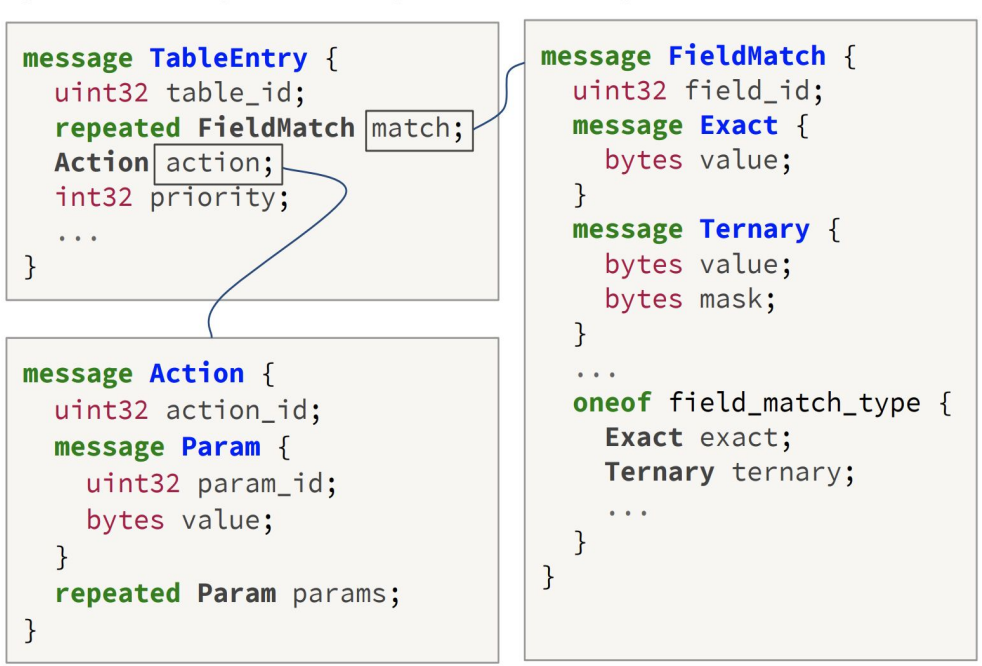
P4Runtime

- API for runtime control for P4-defined data planes
- Based on protobuf and gRPC
  - protobuf: Language for describing data for serialization in a structured way
  - gRPC: Transport over HTTP/2.0 and TLS
- Enables field-reconfigurability
  - Ability to push new P4 program, i.e. re-configure the switch pipeline, without recompiling the switch software stack



P4Runtime message example

- To add a table entry, the control plane needs to know:
  - IDs of P4 entities
    - Tables, field matches, actions, params, etc
  - Field matches for the particular table
    - Match type, bitwidth, etc.
  - Parameters for the particular action
- P4Info is needed by the control plane to format the body of P4Runtime messages (e.g. to add table entry)



P4Info

basic_router.p4

```
...

action ipv4_forward(bit<48> dstAddr,
                    bit<9> port) {
   /* Action implementation */
}

...

table ipv4_lpm {
   key = {
       hdr.ipv4.dstAddr: lpm;
   }
   actions = {
       ipv4_forward;
       ...
   }
   ...
}
```

P4 compiler

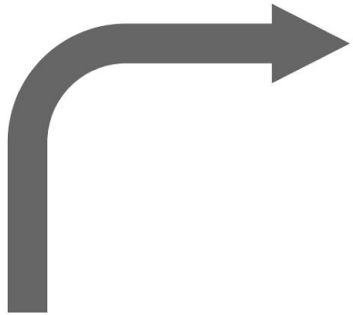
basic_router.p4info

```
actions {
  id: 16786453
  name: "ipv4_forward"
  params {
    id: 1
    name: "dstAddr"
    bitwidth: 48
    ...
    id: 2
    name: "port"
    bitwidth: 9
  }
}
...
tables {
  id: 33581985
  name: "ipv4_lpm"
  match_fields {
    id: 1
    name: "hdr.ipv4.dstAddr"
    bitwidth: 32
    match_type: LPM
  }
  action_ref_id: 16786453
}
```

basic_router.p4

```
action ipv4_forward(bit<48> dstAddr,
                    bit<9>  port) {
    /* Action implementation */
}
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        ...
    }
    ...
}
```

Control plane generates



Protobuf message

```
table_entry {
  table_id: 33581985
  match {
    field_id: 1
    lpm {
      value: "\n\000\001\001"
      prefix_len: 32
    }
  }
  action {
    action_id: 16786453
    params {
      param_id: 1
      value: "\000\000\000\000\000\n"
    }
    params {
      param_id: 2
      value: "\000\007"
    }
  }
}
```

Logical view of table entry

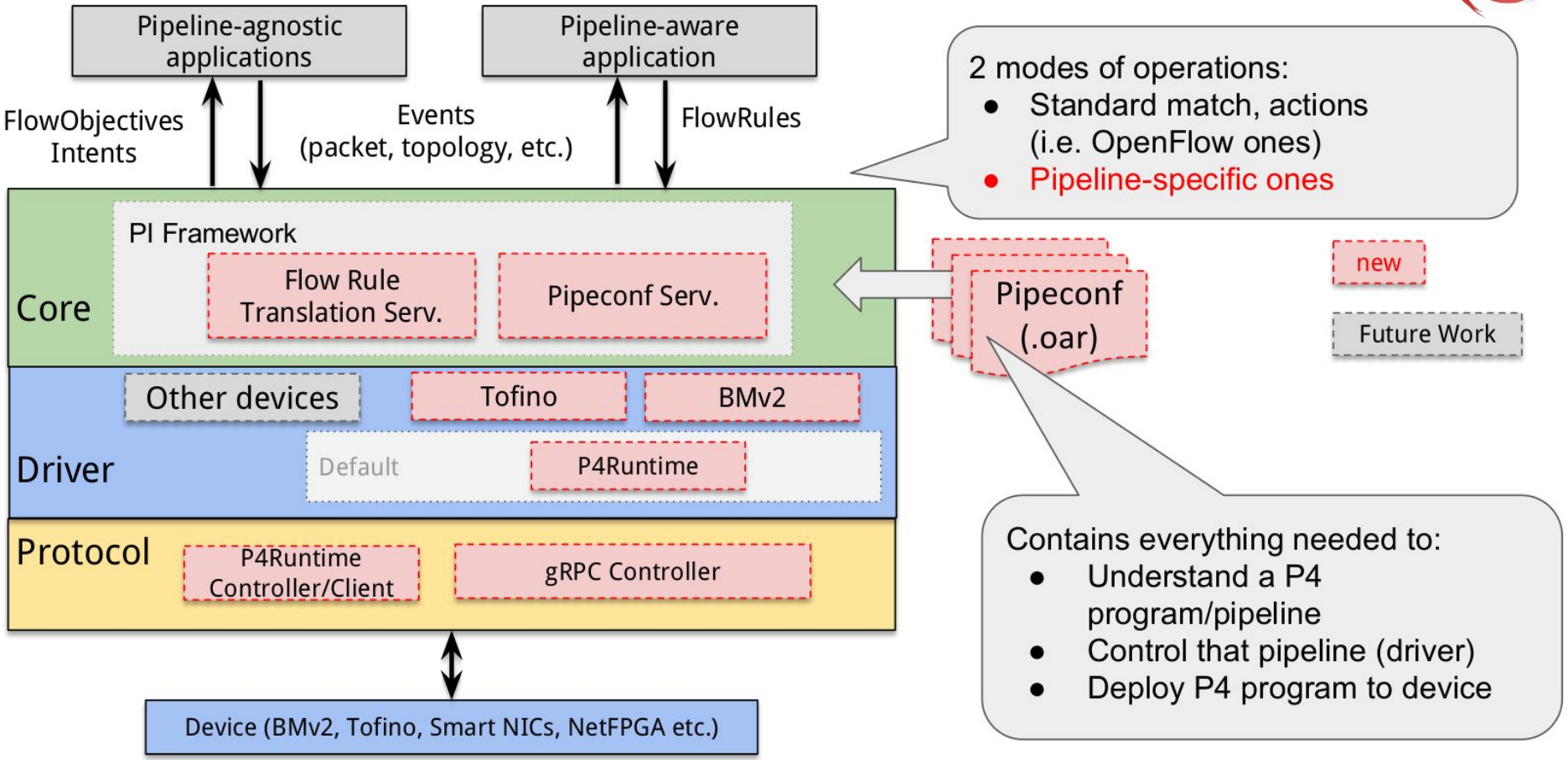
```
hdr.ipv4.dstAddr=10.0.1.1/32
      -> ipv4_forward(00:00:00:00:00:10, 7)
```

P4 and P4 runtime support in ONOS

1. Allow ONOS users to bring their own P4 program
2. Allow apps to control custom / new protocols, as defined in the P4 program
3. Allow existing apps to control any P4 pipeline without changing the app, i.e. enable app portability across many P4 pipelines

P4 and P4 runtime support in ONOS

1. Allow ONOS users to bring their own P4 program
2. Allow apps to control custom/new protocols, as defined in the P4 program
   → pipeconf: let ONOS understand, control, and deploy an arbitrary pipeline
3. Allow existing apps to control any P4 pipeline without changing the app, i.e. enable app portability across many P4 pipelines
   → Translation service


Pipeline-agnostic applications
Pipeline-aware application
FlowObjectives
Intents
Events
(packet, topology, etc.)
FlowRules
Core
PI Framework
Flow Rule Translation Serv.
Pipeconf Serv.
Driver
Other devices
Tofino
BMv2
Default
P4Runtime
Protocol
P4Runtime Controller/Client
gRPC Controller
Device (BMv2, Tofino, Smart NICs, NetFPGA etc.)
2 modes of operations:
Standard match, actions (i.e. OpenFlow ones)
Pipeline-specific ones
Pipeconf (.oar)
new
Future Work
Contains everything needed to:
Understand a P4 program/pipeline
Control that pipeline (driver)
Deploy P4 program to device

P4Runtime support in ONOS 2.1

P4Runtime control entity	ONOS API
Table entry	Flow Rule Service, Flow Objective Service Intent Service
Packet-in/out	Packet Service
Action profile group/members, PRE multicast groups, clone sessions	Group Service
Meter	Meter Service (indirect meters only)
Counters	Flow Rule Service (direct counters) P4Runtime Client (indirect counters)
Pipeline Config	Pipeconf

Unsupported features - community help needed!
Parser value sets, registers, digests

PI Framework

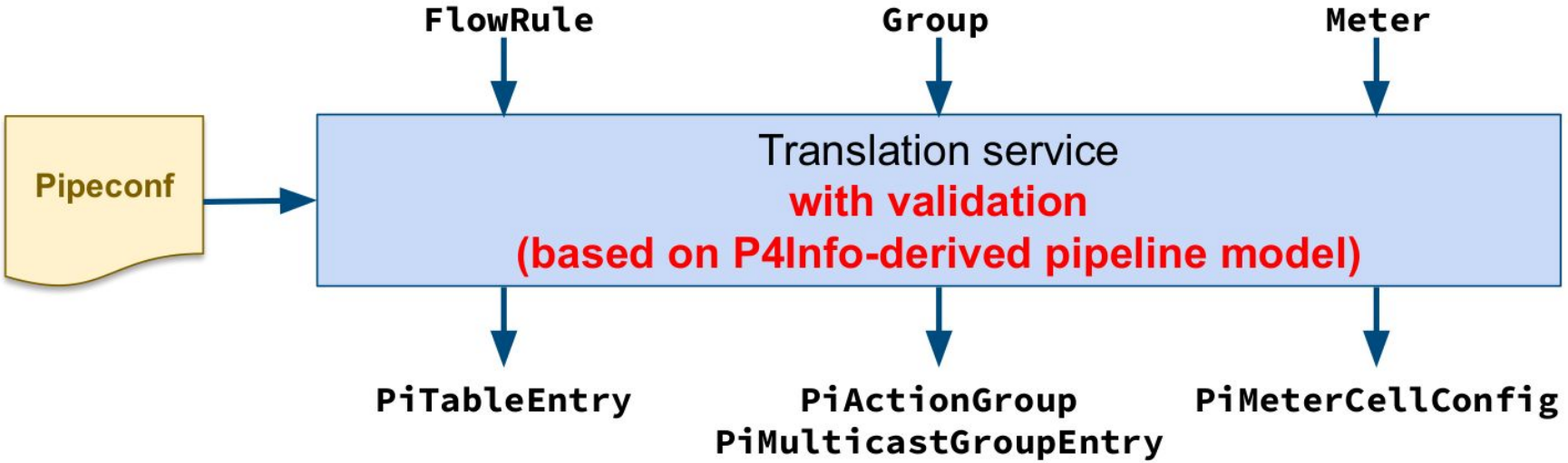
- PI = protocol/program/pipeline independent
- Set of classes and services to aid in the control of programmable data planes
  - All starting with Pi*, e.g. PiPipeconf, PiTableEntry, etc.
- P4 program-specific match fields/actions
  - PiCriterion: specify all field matches using
    - Field name (e.g. my_tunnel_header.tunnel_id)
    - Match type (exact, ternary, LPM)
    - Value/mask (bytes)
  - PiInstruction: wrapper around PI table action
    - Action name (as in the P4 program, e.g. set_agress_port)
    - Action parameters
      - Parameter name (as in the P4 program, e.g. port_id)
      - Parameter value (bytes)

Pipeconf

- Let ONOS understand, control, and deploy an arbitrary pipeline
- Provided to ONOS as applications (.oar)
  - Pipeline Model: Pipeline entities description (i.e. parsed P4 programs)
    - Description of the pipeline understood by ONOS
    - Automatically derived from P4Info
  - Pipeline-specific driver behaviors (e.g. Pipeliner, Interpreter, etc.)
  - Target-specific extensions
    - Compiled P4 program binaries (e.g. for Tofino, BMv2, etc.)
    - P4Info, needed by driver for ID-name mapping
- example

Translation service

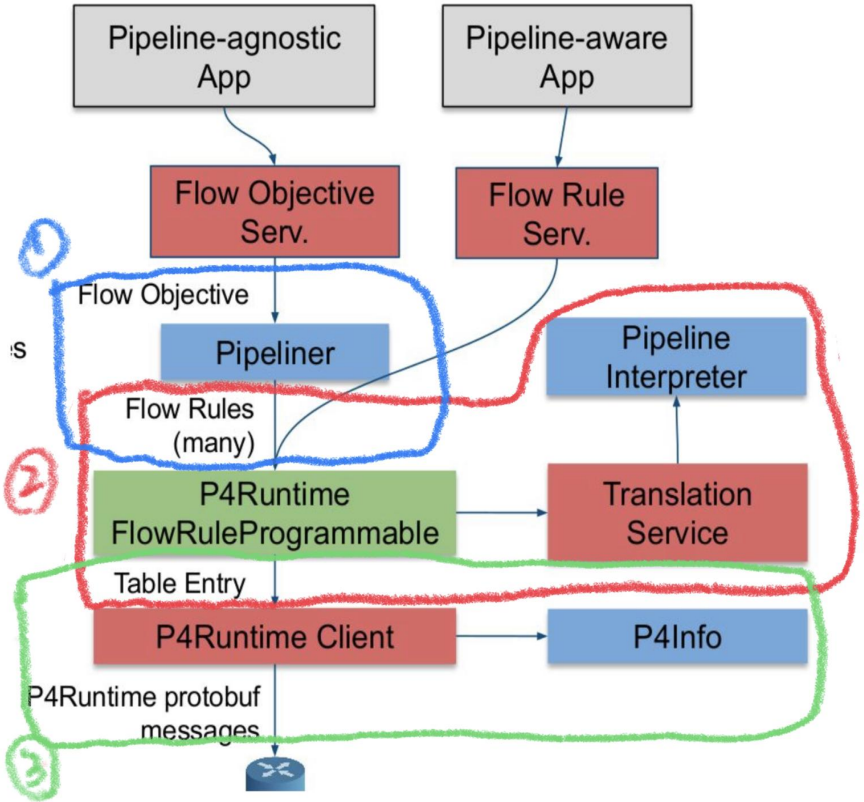
- Translate pipeline-specific entities from protocol-dependent representations to PI ones
  - E.g. OpenFlow-like headers/criteria and actions to P4-specific ones



Flow operations

Pipeconf-based 3-phase translation

1. Flow objective → Flow Rule
   a. Maps 1 flow objective to many flow rules
2. Flow Rule → Table entry
   a. E.g. ETH_DST→“hdr.ethernet.dst_addr”
3. Table entry → P4Runtime message
   a. Maps P4 names to P4Info numeric IDs
   b. “hdr.ethernet.dst_addr” → 3498746



Pipeline Interpreter

- Provide mapping from OpenFlow-derived ONOS headers/actions to P4 program-specific entities
- Example: flow rule translation
  - Match
    - 1:1 mapping between ONOS criteria and P4 header names
    - E.g. ETH_DST → ethernet.dst_addr (name defined in P4 program)
  - Action
    - ONOS defines standard actions as in OpenFlow (output, set field, etc.)
    - P4 allows only one action per table entry, ONOS many (as in OpenFlow)
    - E.g. header rewrite + output: 2 actions in ONOS, 1 action with 2 parameters in P4
    - How to map many actions to one? Need interpretation logic (i.e. Java code)!
- Used also for other purposes
  - Map ONOS table integer IDs to names, packet I/O operations, table counters, etc.
  - packet I/O operations: encapsulation format is defined in the P4 program

ONOS + P4 workflow

- Write P4 program and compile it
  - Obtain P4Info and target-specific binaries to deploy on device
- Create pipeconf
  - Implement pipeline-specific driver behaviours (Java):
    - Pipeliner (optional - if you need FlowObjective mapping)
    - Pipeline Interpreter (to map ONOS known headers/actions to P4 program ones)
    - Other driver behaviors that depend on pipeline
- Use existing pipeline-agnostic apps
  - Apps that program the network using FlowObjectives
- Write new pipeline-aware apps
  - Apps can use same string names of tables, headers, and actions as in the P4 program

netcfg

- This JSON file informs ONOS of the existence of such devices when it is pushed
- Once the device is present in ONOS you can interact with it
- sudo -E mn
  --custom $BMV2_MN_PY
  --switch onosbmv2,
  pipeconf=nctu.pncourse.pipeconf
  --controller remote,ip=127.0.0.1
  --topo=tree,2

Recap

1. Descirbe your P4 pipeline.
2. Describe the behavior added in your pipeconf app and why you add it.
3. Describe the whole workflow of how your p4-learning-bridge app installs the flow rule into the p4 switch.
4. Describe how you implement the flooding function.

Reference

- https://hackmd.io/@cnsrl/H1Yn_xjDU
- https://p4.org/assets/P4_D2_East_2018_02_p4runtime.pdf
- https://github.com/p4lang/behavioral-model