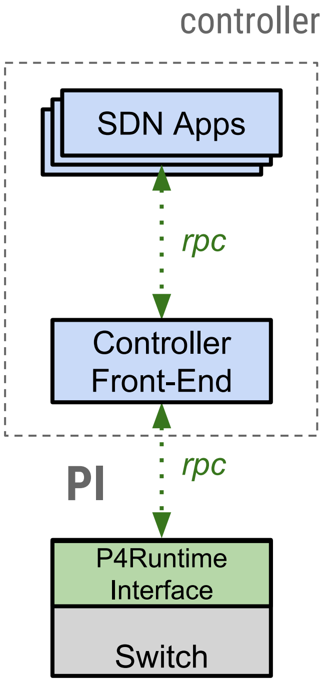
P4 Program-Dependent Controller Interface for SDN Applications

P4 Workshop 2017

Samar Abdi, Waqar Mohsin, Yavuz Yetim, Alireza Ghaffarkhah

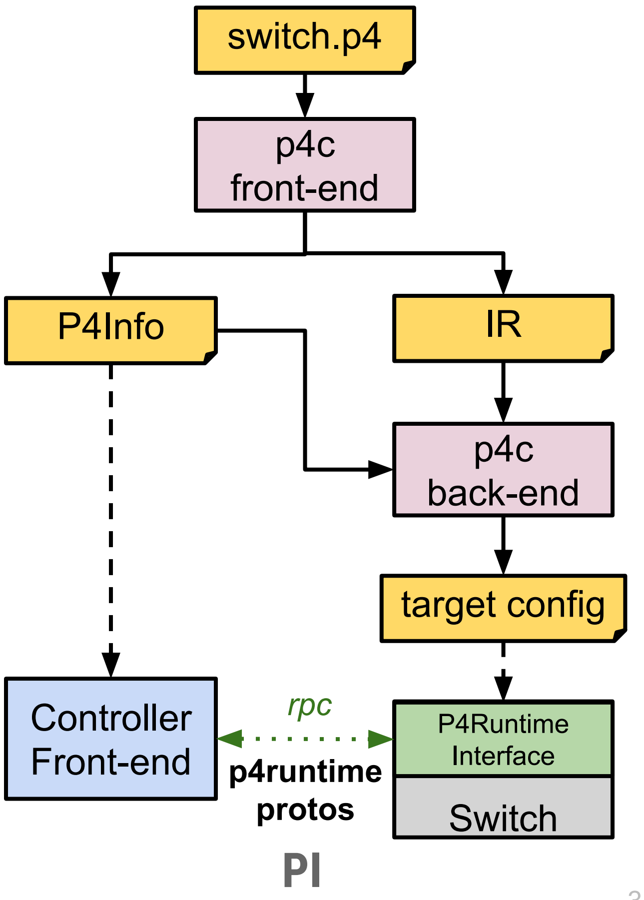
Background

- P4 offers a **formal contract** between controller and switch
  - Controller's logical view of forwarding is implemented by the switch
  - Enables silicon independence
- P4 API WG proposes a **P4Runtime** switch interface
  - A runtime API to manage P4 table entries
- P4Runtime is **PI (program independent)**
  - API (message definitions, RPCs) doesn't change if P4 program changes
- Appeal of a PI runtime interface
  - Stable API for easier vendor adoption
  - Enables field reconfigurability
    - ability to push new P4 program without recompiling deployed switches



P4Runtime PI Interface Workflow

- P4Info proto
  - captures target-independent P4 program attributes
  - defines IDs for P4 tables, actions, params, etc.
- IR = P4 compiler *intermediate representation*
- Target Config
  - P4Info + P4-program mapping to silicon
- P4Runtime defines the PI interface
  - Refers to P4 entities by integer IDs coming from P4Info

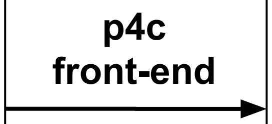


Google

P4Info Example

```
action set_vrf(bit<32> id) {
  meta.vrf_id = id;
}
table vrf_classifier_table {
  key = {
    hdr.ethernet.etherType : exact;
    hdr.ethernet.srcAddr : ternary;
    smeta.ingress_port: exact;
  }
  actions = {
    set_vrf;
  }
  default_action = set_vrf(0);
}
```

vrf.p4




```
action id: 16777233
  param id: 50336000

table id: 33554433
  match_field id: 67108875
    match_type: EXACT
  match_field id: 67108864
    match_type: TERNARY
  match_field id: 67108870
    match_type: EXACT

  action_ref id: 16777233
```

P4Info for vrf.p4

Google

PI Proto Example

```
action set_vrf(bit<32> id) {
  meta.vrf_id = id;
}
table vrf_classifier_table {
  key = {
    hdr.ethernet.etherType : exact;
    hdr.ethernet.srcAddr : ternary;
    smeta.ingress_port: exact;
  }
  actions = {
    set_vrf;
  }
  default_action = set_vrf(0);
}
```

vrf.p4

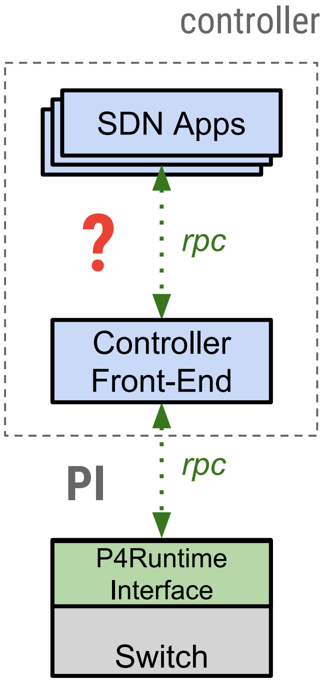
```
table_entry {
  table_id: 33554433
  match {
    field_id: 67108875
    exact {
      value: \x08\x00
    }
  }
  match {
    field_id: 67108870
    exact {
      value:
\x00\x00\x00\x00\x11\x01
    }
  } ...
```

```
...
  table_action {
    action {
      action_id: 16777233
      params {
        param_id: 50336000
        value:
\x00\x00\x00\x70
      }
    }
  }
}
```

PI message instance

But what about API exposed to SDN apps?

- A program-independent (PI) API seems a poor fit for **direct use** by SDN apps. Concerns ...
  - Readability
  - Type safety
  - Versioning friendliness (backwards compatibility)



Google

PI Proto Example

```
action set_vrf(bit<32> id) {
  meta.vrf_id = id;
}
table vrf_classifier_table {
  key = {
    hdr.ethernet.etherType : exact;
    hdr.ethernet.srcAddr : ternary;
    smeta.ingress_port: exact;
  }
  actions = {
    set_vrf;
  }
  default_action = set_vrf(0);
}
```

```
table_entry {
  table_id: 33554433
  match {
    field_id: 67108875
    exact {
      value: \x08\x00
    }
  }
  match {
    field_id: 67108870
    exact {
      value:
\x00\x00\x00\x00\x11\x01
    }
  } ...
```

```
...
  table_action {
    action {
      action_id: 16777233
      params {
        param_id: 50336000
        value:
\x00\x00\x00\x70
      }
    }
  }
}
```

- Difficult for SDN app to directly populate this proto
- Untyped data (bytes)

P4 Program Evolution

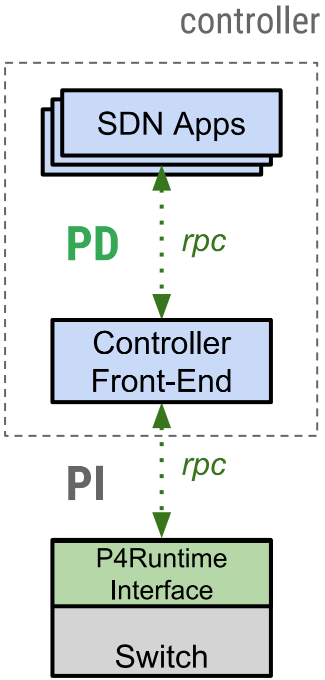
	P4 Source	PI Proto
Version 1	table vrf_classifier_table { key = { ... hdr.ethernet.srcAddr : ternary; } ... }	table_entry { table_id: **33554433** match { field_id: **67108870** exact { value: *37* } …
Version 2	table vrf_classifier_table { key = { ... *// deprecated* hdr.ethernet.srcAddr : ternary; } ... }	table_entry { table_id: **33554433** ... }

- Version 1 PI proto cannot be processed by Version 2 tools
  - field_id: *67108864* not recognized

But what about API exposed to SDN apps?

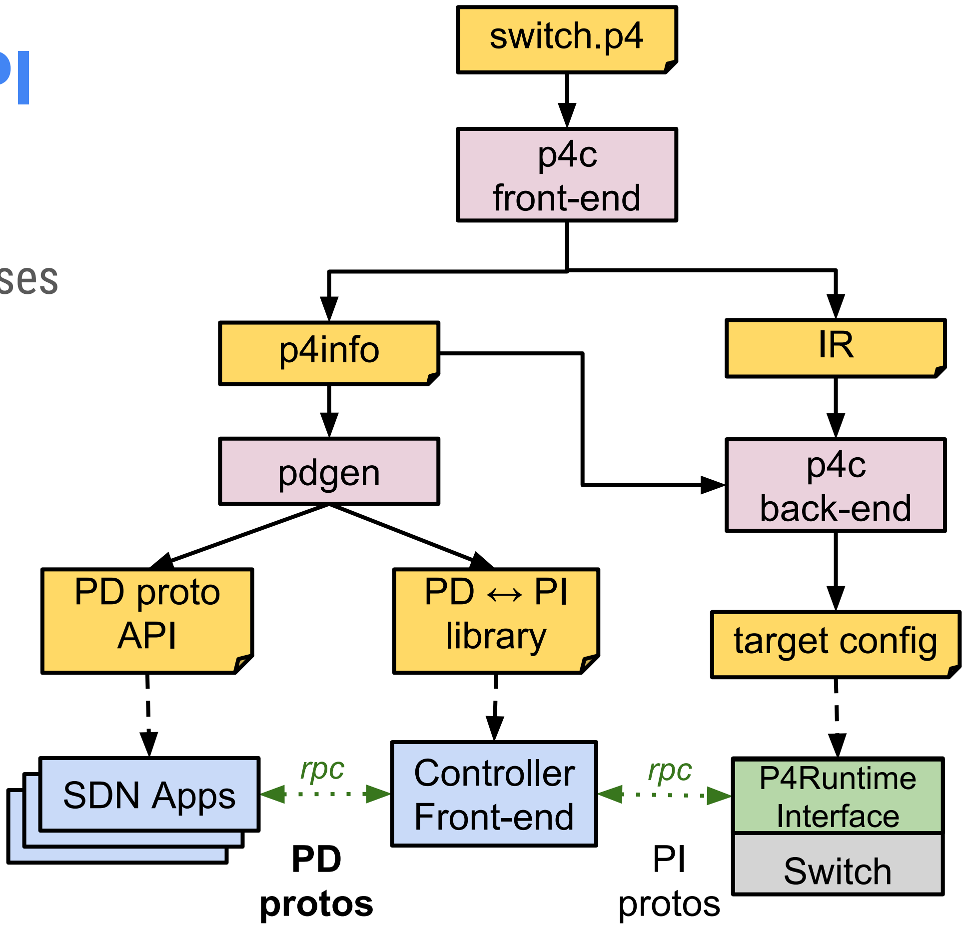
- A program-independent (PI) API seems a poor fit for this
  - Readability
  - Type safety
  - Versioning friendliness (backwards compatibility)

- **Proposal**: A **Program-Dependent (PD) API** to address these concerns
  - Defined using Protocol Buffers
  - Tailored API (message definitions) that changes with P4 program
  - Auto-generated library to translate to/from PI messages



PD (Program-Dependent) API

- PD protos capture flow messages/responses
  - Refer to P4 tables by name
  - Strict typing of match fields
- PD benefits
  - Improved readability and type safety (typed-proto vs bytes)
  - Tooling-friendliness: materialized protos can be analyzed
  - Upgrade/Downgrade and versioning friendly (typed-proto characteristics)



Google

P4 to PD Proto Encoding Rules

- PD protos generated for table entries, actions and action profile members
- Type conversion
  - P4 bit-vectors converted to proto ‘uint’s or bytes
  - Translation library performs runtime validation of bitwidth

P4 field/mask bitwidth	**Proto type**		
	exact	**ternary**	**lpm**
1 <= bitwidth <= 32	uint32	Ternary32	LPM32
32 < bitwidth <= 64	uint64	Ternary64	LPM64
bitwidth > 64	bytes	Ternary	LPM

Ternary32	uint32 mask
	uint32 value
LPM64	uint32 value
	uint32 prefix_len

PD Proto Schema Example

```
action set_vrf(bit<32> id) {
  meta.vrf_id = id;
}
table vrf_classifier_table {
  key = {
    hdr.ethernet.ether_type : exact;
    hdr.ethernet.src_addr : ternary;
    smeta.ingress_port: exact;
  }
  actions = {
    set_vrf;
  }
  default_action = set_vrf(0);
}
```

vrf.p4

pdgen →

```
package p4.vrf;
message SetVrfAction {
  uint32 vrf_id = 1;
}
message VrfClassifierTableEntry {
  message Match {
    uint32     ethernet_ether_type = 1;
    Ternary64  ethernet_src_addr = 2;
    uint32     smeta_ingress_port = 3;
  }
  Match match = 1;
  message Action {
    SetVrfAction set_vrf = 1;
  }
  Action action = 2;
}
```

PD proto for vrf.p4

Table Entry Generation Example

```
package p4.vrf;
message SetVrfAction {
  uint32 vrf_id = 1;
}
message VrfClassifierTableEntry {
  message Match {
    uint32    ethernet_ether_type = 1;
    Ternary64 ethernet_src_addr = 2;
    uint32    smeta_ingress_port = 3;
  }
  Match match = 1;
  message Action {
    SetVrfAction set_vrf = 1;
  }
  Action action = 2;
}
```

PD proto for vrf.p4

```
VrfClassifierTableEntry table_entry;
VrfClassifierTableEntry::Match *match =
              table_entry.mutable_match();

match->set_ethernet_ether_type(0x0800);
match->set_smeta_ingress_port(37);

VrfClassifierTableEntry::Action *action =
              table_entry.mutable_action();

action->set_vrf->set_vrf_id(112);
```

SDN app code (C++)

Google

P4 Program Evolution

	P4 Source	PD Proto
Version 1	`table vrf_classifier_table {` `  key = {` `    ...` `    hdr.ethernet.srcAddr : ternary @tag(2);` `  } ...` `}`	`message VrfClassifierTableEntry {` `  message Match {` `    ...` `    Ternary64 ethernet_src_addr = 2;` `  } ...` `}`

P4 Program Evolution

	P4 Source	PD Proto
Version 1	`table vrf_classifier_table {` `  key = {` `    ...` `    hdr.ethernet.srcAddr : ternary @tag(2);` `  } ...` `}`	`message VrfClassifierTableEntry {` `  message Match {` `    ...` `    Ternary64 ethernet_src_addr = 2;` `  } ...` `}`
Version 2	`@deprecated_tag("hdr.ethernet.srcAddr : ternary",2);` `table vrf_classifier_table {` `  key = {` `    ...` `  } ...` `}`	`message VrfClassifierTableEntry {` `  message Match {` `    Ternary64 hdr_bar = 2 [deprecated = true];` `  } ...` `}`

- Version 1 PD proto **can** be processed by Version 2 tools
- Version 1 controller code will still compile with Version 2 switch: enables staging

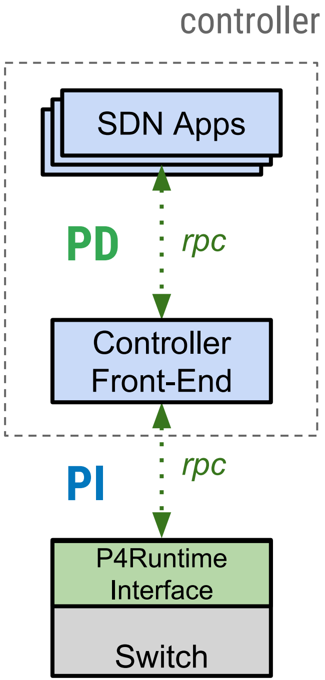
Google

P4 Program Evolution

<table>
<tr><th></th><th>P4 Source</th><th>PD Proto</th></tr>
<tr><td>Version 1</td><td><pre>table vrf_classifier_table {
  key = {
    ...
    hdr.ethernet.srcAddr : ternary @tag(2);
  } ...
}</pre></td><td><pre>message VrfClassifierTableEntry {
  message Match {
    ...
    Ternary64 ethernet_src_addr = 2;
  } ...
}</pre></td></tr>
<tr><td>Version 2</td><td><pre>@deprecated_tag("hdr.ethernet.srcAddr : ternary",2);
table vrf_classifier_table {
  key = {
    ...
  } ...
}</pre></td><td><pre>message VrfClassifierTableEntry {
  message Match {
    Ternary64 hdr_bar = 2 [deprecated = true];
  } ...
}</pre></td></tr>
<tr><td>Version N</td><td><pre>@reserved_tag(2)
table vrf_classifier_table {
  key = {
    ...
  } ...
}</pre></td><td><pre>message VrfClassifierTableEntry {
  message Match {
    reserved 2;
  } ...
}</pre></td></tr>
</table>

Conclusion

- PI interface between controller front-end and switch
  - supports field reconfigurability and easier vendor adoption
- PD interface between SDN apps and controller front-end
  - Type safe, tooling-friendly and versioning-friendly
- Toolchain support
  - Protocol buffers for communicating PD and PI messages
  - Auto-generation of PD proto schema from P4 program
  - Auto-generation of PD↔PI translation library



Google