Tutorial:
P4 and P4Runtime Technical Introduction and Use Cases for Service Providers

Carmelo Cascone
Open Networking Foundation

Open Networking Summit 2018, September 27 2018
Outline

● P4
● P4Runtime
● ONOS support for P4/P4Runtime
● Use cases
  ○ Vendor/silicon-independent fabric
  ○ VNF offloading
P4
The Data Plane Programming Language
P4 - The Data Plane Programming Language

- Domain-specific language to formally define the data plane pipeline behavior
  - Describe protocol headers, lookup tables, actions, counters, etc.
  - Can describe fast pipelines (e.g. ASIC, FPGA) as well as a slower ones (e.g. SW switch)

- Good for programmable switches, as well as fixed-function ones
  - Defines “contract” between the control plane and data plane for runtime control

```
Packets

Table {
  match
  actions
}
```

```
mypipeline.p4
```

- Compiler (provided by switch vendor)
  - Configure programmable ASIC/FPGA
  - or maps to fixed-function ASIC tables

```
Programmable or fixed-function data plane pipeline
```

Copyright © 2018 - Open Networking Foundation
Evolution of the language

- **P4\textsubscript{14}**
  - Original version of the language
  - Assumed specific device capabilities
  - Good only for a subset of programmable switch/targets

- **P4\textsubscript{16}**
  - More mature and stable language definition
  - Does not assume device capabilities, which instead are defined by target manufacturer via external libraries/architecture definition
  - Good for many targets, e.g. switches and NICS, programmable or fixed-function
  - Focus of this tutorial
PISA: Protocol-Independent Switch Architecture

Abstract machine model of programmable switch architecture

Programmer declares the headers that should be recognized and their order in the packet

Programmer defines the tables and the exact processing algorithm

Programmer declares how the output packet will look on the wire
Mapping a simple logical pipeline on PISA

Large
IPv4
Small
IPv6
ACL

P4 compiler
Allocate resources to realize the pipeline

Programmable Parser
Programmable Match-Action Pipeline
Programmable Deparser

Ethernet address table
IPv4 address table
IPv4 address table
IPv6 address table
ACL filter table

Allocate resources to realize the pipeline
P4 programs and architectures

my_program.p4
Defines the processing of each block

architecture.p4
Defines which blocks are available, the interfaces of each block, and their capabilities
PSA - Portable Switch Architecture

- Community-developed architecture
  - [https://github.com/p4lang/p4-spec/tree/master/p4-16/psa](https://github.com/p4lang/p4-spec/tree/master/p4-16/psa)
- Describes common capabilities of a network switch
  - Which process and forward packets across multiple interface ports
- 6 programmable P4 blocks + 2 fixed-function blocks
- Defines capabilities beyond match+action tables
  - Counters, meters, stateful registers, hash functions, etc.
```c
#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;
```
header ethernet_t {
    bit<48> dst_addr;
    bit<48> src_addr;
    bit<16> eth_type;
}

header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    ...
}

parser parser_impl(packet_in pkt, out headers_t hdr) {
    /* Parser state machine to extract header fields */
}

action set_next_hop(bit<48> dst_addr) {
    ethernet.dst_addr = dst_addr;
    ipv4.ttl = ipv4.ttl - 1;
}

... 

table ipv4_routing_table {
    key = {
        ipv4.dst_addr : LPM; // longest-prefix match
    }
    actions = {
        set_next_hop();
        drop();
    }
    size = 4096; // table entries
}
Simple router example

- **Data plane (P4) program**
  - Defines the format of the table
    - Match fields, actions, action data (parameters)
  - Performs the lookup
  - Executes the chosen action

- **Control plane**
  - Populates table entries with specific information
    - Based on configuration, automatic discovery, protocol calculations

```c
action ipv4_forward(bit<48> dst_addr, bit<9> port) {
    ethernet.dst_addr = dst_addr;
    standard_metadata.egress_spec = port;
    ipv4.ttl = ipv4.ttl - 1;
}

table ipv4_routing_table {
    key = {
        ipv4.dst_addr : LPM;  // longest-prefix match
    }
    actions = {
        ipv4_forward();
        drop();
    }
}
```

### Data plane (P4) program

- Defines the format of the table
- Performs the lookup
- Executes the chosen action

### Control plane

- Populates table entries with specific information
- Based on configuration, automatic discovery, protocol calculations

---

**Control plane populates table entries**

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
<th>Action Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.1.1</td>
<td>ipv4_forward</td>
<td>dstAddr=00:00:00:00:01:01</td>
</tr>
<tr>
<td>10.0.1.2</td>
<td>drop</td>
<td>port=1</td>
</tr>
<tr>
<td>*</td>
<td>NoAction</td>
<td></td>
</tr>
</tbody>
</table>
P4 workflow

1. **P4 Program**
2. **P4 Compiler**
3. **Target-specific configuration binary**
4. **Load**
5. **Control Plane**
   - Add/remove table entries
   - Extern control
   - Packet-in/out
   - CPU port
6. **Data Plane**
   - Tables
   - Extern objects
7. **Run-time control**

**User supplied**

**Vendor supplied**
P4Runtime
Control protocol for P4-defined data planes
Traditional/OpenFlow vs. P4 paradigm

Traditional switch
OpenFlow or legacy

Control plane
Table mgmt
Control traffic

P4-runtime
P4 table mgmt

P4-defined switch

Data plane
Packets

Slide courtesy P4.org
Do we need yet another data plane control API?

**How Standards Proliferate:**

(see: A/C chargers, character encodings, instant messaging, etc.)

**Situation:**

There are 14 competing standards.

**14?! Ridiculous!**

We need to develop one universal standard that covers everyone's use cases. **Yeah!**

**Soon:**

**Situation:**

There are 15 competing standards.
Yes, we need P4Runtime

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

- P4 program tailored to apps / role - does not describe the hardware
- Switch maps program to fixed-function ASIC
- Enables portability
P4Runtime overview

- Protocol for runtime control of P4-defined switches
  - Designed around PSA architecture but can be extended to others

- Work-in-progress by the p4.org API WG
  - Initial contribution by Google and Barefoot
  - Draft of version 1.0 available: https://p4.org/p4-spec/

- Protobuf-based API definition
  - Automatically generate client/server code for many languages
  - gRPC transport

- P4 program-independent
  - API doesn’t change with the P4 program

- Enables field-reconfigurability
  - Ability to push new P4 program, i.e. re-configure the switch pipeline, without recompiling the switch software stack
Protocol Buffers (protobuf) Basics

- Language for describing data for serialization in a structured way
- Common binary wire-format
- Language-neutral
  - Code generators for: Action Script, C, C++, C#, Clojure, Lisp, D, Dart, Erlang, Go, Haskell, Java, Javascript, Lua, Objective C, OCaml, Perl, PHP, Python, Ruby, Rust, Scala, Swift, Visual Basic, ...
- Platform-neutral
- Extensible and backwards compatible
- Strongly typed

syntax = "proto3";

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }

  repeated PhoneNumber phone = 4;
}
gRPC Basics

- Use Protobuf to define service API and messages
- Automatically generate native stubs in:
  - C / C++, C#, Dart, Go, Java, Node.js, PHP, Python, Ruby
- Transport over HTTP/2.0 and TLS
  - Efficient single TCP connection implementation that supports bidirectional streaming
p4runtime.proto (gRPC service)

Enables a local or remote entity to load the pipeline/P4 program, send/receive packets, arbitrate mastership, read and write forwarding table entries, counters, and other P4 entities.

```protobuf
service P4Runtime {
  rpc Write(WriteRequest) returns (WriteResponse) {}  
  rpc Read(ReadRequest) returns (stream ReadResponse) {}  
  rpc SetForwardingPipelineConfig(SetForwardingPipelineConfigRequest)
      returns (SetForwardingPipelineConfigResponse) {}  
  rpc GetForwardingPipelineConfig(GetForwardingPipelineConfigRequest)
      returns (GetForwardingPipelineConfigResponse) {}  
  rpc StreamChannel(stream StreamMessageRequest)
      returns (stream StreamMessageResponse) {}  
}
```

message WriteRequest {
    uint64 device_id = 1;
    uint64 role_id = 2;
    Uint128 election_id = 3;
    repeated Update updates = 4;
}

message Update {
    enum Type {
        UNSPECIFIED = 0;
        INSERT = 1;
        MODIFY = 2;
        DELETE = 3;
    }
    Type type = 1;
    Entity entity = 2;
}

message Entity {
    oneof entity {
        ExternEntry extern_entry = 1;
        TableEntry table_entry = 2;
        ActionProfileMember
            action_profile_member = 3;
        ActionProfileGroup
            action_profile_group = 4;
        MeterEntry meter_entry = 5;
        DirectMeterEntry direct_meter_entry = 6;
        CounterEntry counter_entry = 7;
        DirectCounterEntry direct_counter_entry = 8;
        PacketReplicationEngineEntry
            packet_replication_engine_entry = 9;
        ValueSetEntry value_set_entry = 10;
        RegisterEntry register_entry = 11;
    }
}

Copyright © 2018 - Open Networking Foundation
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

- Other P4 program attributes
P4 compiler workflow

P4 compiler generates 2 files:

1. **Target-specific binaries**
   - Used to configure switch pipeline
     - (e.g. binary config for ASIC, bitstream for FPGA, etc.)

2. **P4Info file**
   - Describes “schema” of pipeline for runtime control
   - Captures P4 program attributes
     - Tables, actions, parameters, etc.
   - Protobuf-based format
   - Target-independent compiler output
     - Same P4Info for SW switch, ASIC, etc.

Full P4Info protobuf specification:
https://github.com/p4lang/p4runtime/blob/master/proto/p4/config/v1/p4info.proto
P4Info example

basic_router.p4

```p4
... action ipv4_forward(bit<48> dstAddr, bit<9> port) {
  /* Action implementation */
}
...

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

basic_router.p4info

```p4
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
}
```
P4Runtime table entry example

---

**basic_router.p4**

```p4
action ipv4_forward(bit<48> dstAddr, bit<9> port) {
    /* Action implementation */
}
```

```p4
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        ...
    }
    ...
}
```

**Protobuf message**

```p4
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\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)`

**Control plane generates**
**P4Runtime SetPipelineConfig**

```plaintext
message SetForwardingPipelineConfigRequest {
  enum Action {
    UNSPECIFIED = 0;
    VERIFY = 1;
    VERIFY_AND_SAVE = 2;
    VERIFY_AND_COMMIT = 3;
    COMMIT = 4;
    RECONCILE_AND_COMMIT = 5;
  }
  uint64 device_id = 1;
  uint64 role_id = 2;
  Uint128 election_id = 3;
  Action action = 4;
  ForwardingPipelineConfig config = 5;
}
```

```plaintext
message ForwardingPipelineConfig {
  config.P4Info p4info = 1;
  // Target-specific P4 configuration.
  bytes p4_device_config = 2;
}
```
Project Stratum - P4Runtime switch agent implementation

- Open source, lightweight, production quality thin switch OS
- Implements next-gen SDN interfaces
  - P4Runtime for control
    - Uses P4 as the data pipeline contract across fixed function and programmable hardware
  - gNMI using OpenConfig models for configuration/monitoring/telemetry
  - gNOI for operations
- Rich community of service and cloud providers, chipset vendors, whitebox and blackbox switch vendors
  - Google committed to using Stratum in production network at scale

https://stratumproject.org/
ONOS
A control plane for P4/P4Runtime devices
ONOS architecture recap

Control and configure the network using a **global topology view** and independently of the device-specific details.

**Northbound API**
- Topology API
- FlowRule API
- FlowObjective API
- Intent API
- Packet API

**Device driver**
Allow device-specific variants of standard protocols.

**Protocol libraries**
- OpenFlow
- P4Runtime
- Netconf

**Distributed core**
State management, notifications, high-availability & scale-out

- OVS
- Arista
- Barefoot
- Mellanox
- Ciena
- Cisco
- Corsa
- Fujitsu
- HP
- Huawei
- Juniper
- Lumentum
- Microsemi
- Polatis
- ...more
P4 and P4Runtime support in ONOS

Goals:

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 accros many P4 pipelines
Pipeline-aware/agnostic apps

Pipeline-agnostic app
- P4Runtime server (e.g. Stratum)
- Pipeline-specific Flow Rules, Groups, Meters, etc

Translation
Maps FlowObjective to pipeline-specific table entries

Pipeline-aware app
- Events (packet, topology, etc.)

ONOS
- FlowObjective API
- Core
- Events
- Device drivers
- Protocol libraries
- P4Runtime client
- gRPC
  Deploy P4 program
  Table management, etc
- Switch
  P4Runtime server (e.g. Stratum)

Pipeconf Store
- P4 pipeline
- Contains:
  - P4Info (pipeline model)
  - P4 compiler output, to program device pipeline
  - “Pipeline drivers” used by ONOS translation services

Define flow rules using same table names and headers/action as in the P4 program. E.g match on “hdr.my_protocol.my_field”

Powerful ONOS API, allows apps to declare device-level forwarding intents
## P4Runtime support in ONOS 1.14 (Owl)

<table>
<thead>
<tr>
<th>P4Runtime control entity</th>
<th>ONOS API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table entry</td>
<td>Flow Rule Service, Flow Objective Service Intent Service</td>
</tr>
<tr>
<td>Packet-in/out</td>
<td>Packet Service</td>
</tr>
<tr>
<td>Action profile group/members, PRE multicast</td>
<td>Group Service</td>
</tr>
<tr>
<td>groups</td>
<td></td>
</tr>
<tr>
<td>Meter</td>
<td>Meter Service (indirect meters only)</td>
</tr>
<tr>
<td>Counters</td>
<td>Flow Rule Service (direct counters)</td>
</tr>
<tr>
<td></td>
<td>P4Runtime Client (indirect counters)</td>
</tr>
<tr>
<td>Pipeline Config</td>
<td>Pipeconf</td>
</tr>
</tbody>
</table>

### Unsupported features - community help needed!
Parser value sets, registers, digests, clone sessions
Use case 1: silicon-independent fabric
Trellis – Multi-purpose Leaf-Spine Fabric

● Prominent example of ONOS application
  ○ In production at Comcast

● Multi-purpose leaf-spine fabric designed for NFV and access/edge applications
  ○ Built with white-box switches, open source software, SDN based

● Extensive feature set
  ○ Bridging/VLANs, IPv4/v6 unicast and multicast routing, DHCP-relay, pseudowires, QinQ, vRouter & more

● Works with OpenFlow and P4/P4Runtime
2-stage leaf-spine fabric with multi-vendor white-box switches:
- OFDPA (Broadcom Trident2/Tomahawk)
- Stratum (Barefoot Tofino)
Trellis & P4

Pipeline-agnostic apps - use ONOS FlowObjective API

Trellis apps

- Segment Routing
- DHCP L3 Relay
- vRouter
- Multicast
- ...

ONOS

OF-DPA driver

P4Runtime

fabric.p4 driver

OpenFlow

White-box switches

Broadcom Qumran

Broadcom Tomahawk

Broadcom Trident2

Barefoot Tofino

Mellanox Spectrum 1

Fabric.p4 pipeconf

Copyright © 2018 - Open Networking Foundation
fabric.p4

- **P4 implementation of the Trellis reference pipeline**
  - Inspired by Broadcom OF-DPA pipeline
  - Tailored to Trellis needs (fewer tables, easier to control)
  - Work in progress:
    - Missing support for IPv6, double-VLAN termination

- **Bring more heterogeneity in Trellis with P4-capable silicon**
  - Works with both programmable and fixed-function chips (logical pipeline of legacy L2/L3/MPLS features)
  - Any switch pipeline that can be mapped to fabric.p4 can be used with Trellis

- **Extensible open-source implementation**
fabric.p4 pipeline

In-port + VLAN filtering table

Forwarding classifier

Bridging  MPLS  IPv4 unicast routing  IPv6 unicast routing (WIP)

ACL

Next ID mapping

Unicast  Hashed (ECMP)  Multicast
Use case 2:
VNF offloading
VNF offloading

- Programmable data planes offer great flexibility beyond “plumbing”

<table>
<thead>
<tr>
<th>Progr. ASIC capabilities</th>
<th>VNF building blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary header parsing/deparsing</td>
<td>Domain specific encap/decap (e.g. PPPoE termination, GTP, etc.)</td>
</tr>
<tr>
<td>Stateful memories</td>
<td>TCP connection tracking (L4 load balancing, NAT, firewall, etc.)</td>
</tr>
<tr>
<td>Computational capabilities</td>
<td>Billing</td>
</tr>
</tbody>
</table>

- Benefits
  - **Performance** - VNFs executed at line rate, e.g. O(Tbit/s) for DC switch
  - **Low latency and jitter** - Avoid non-determinism of x86 processing
  - **Power consumption** - Less CPU resources for packet processing, use switch that is there anyways
M-CORD with P4 fastpath

GTP termination implemented with P4 and executed directly on the switch ASIC (spgw.p4)

CORD controller (XOS)
- Trellis Apps
- SPGW-u App

ONOS

P4Runtime
P4 program deployment and table management

Upstream router

Control plane VNFs

Demo @ MWC & ONS NA ‘18

Copyright © 2018 - Open Networking Foundation
PoC P4 implementation of the Serving and Packet Gateway (S/PGW) user plane:
  - ~300 lines of P4_16 code
  - Integrated with fabric.p4
  - [https://github.com/opennetworkinglab/onos/…/spgw.p4](https://github.com/opennetworkinglab/onos/…/spgw.p4)

Good enough to demonstrate end-to-end connectivity
  - Support GTP encap/decap, filtering, charging functionalities

Missing features (future work - need help)
  - QoS, downlink buffering during handovers
SPGW-u ONOS App

3GPP Control and User Plane Separation (CUPS) protocol
Create/modify/delete GTP sessions

SPGW-u App

Trellis Apps

ONOS

Pipeline-aware app
Works only with spgw.p4

P4Runtime
spgw.p4 table entries

fabric+spgw.p4

Spine

Spine

ToR

ToR

3GPP

SPGW-c

MME

HSS

Open source EPC from Intel/Sprint

fabric.p4

Copyright © 2018 - Open Networking Foundation
Residential service edge/BNG (se.p4)

- ONF is working with Deutsche Telekom to open-source a production-grade implementation of a residential service edge/BNG in P4
- Enables fast path for residential access
- Features:
  - PPPoE termination
  - Reverse-path filtering (MAC, IPv4/v6)
  - Metering
  - TR-101 double-VLAN termination
  - 2-label MPLS termination
- Community help needed for integration with Trellis and fabric.p4
Pointers

- P4_16 / P4Runtime specifications
  - [https://p4.org/specs/](https://p4.org/specs/)

- Stratum project
  - [https://stratumproject.org/](https://stratumproject.org/)

- ONOS
  - [https://wiki.onosproject.org/display/ONOS/Wiki+Home](https://wiki.onosproject.org/display/ONOS/Wiki+Home)

- Fabric.p4
  - [https://wiki.onosproject.org/x/wgBkAQ](https://wiki.onosproject.org/x/wgBkAQ)

- Hands-on tutorial with P4/P4Runtime/ONOS
ONOS-P4 Brigade - Join the effort!

Learn more - P4 Brigade Wiki:
https://wiki.onosproject.org/display/ONOS/P4+brigade

P4 Brigade mailing list:
brigade-p4@onosproject.org
Thanks
Silicon-independent remote control

- **Remote control plane (e.g. ONOS/ODL)**
  - OSPF
  - BGP
  - P4-defined custom protocol
  - etc.

- **Target-independent protobuf format**
  
  ```
  table_entry {
    table_id: 33581985
    match {
      field_id: 1
      lpm {
        value: \\f\000\...
        prefix_len: 8
      }
    }
    action {
      action_id: 16786453
      params {
        param_id: 1
        value: \\000\0...
      }
      params {
        param_id: 2
        value: 7
      }
    }
  }
  ```

- **Vendor A (programmable)**
  - P4Runtime agent (e.g. Stratum)
  - Target driver

- **Vendor B (fixed-function)**
  - P4Runtime agent (e.g. Stratum)
  - Target driver

- **Vendor C (fixed-function)**
  - P4Runtime agent (e.g. Stratum)
  - Target driver

- Slide courtesy P4.org
Portability of local control plane

The P4 Runtime API can be used equally well by a remote or local control plane.
Overhead due to GTP and INT headers (when processing small packets)

Inband network telemetry (INT) used to measure GTP processing latency

~490ns to perform GTP encap plus forwarding (ToR 1)

~480ns to perform forwarding (ToR 2)