intel® connectivity academy

by P4Ica
Control Plane APIs: The dark side of P4 Programming

Vladimir Gurevich, P4ica

November 15, 2023
Agenda

- What are Control Plane APIs and why do we need them?
- How can we communicate with a data plane program?
- How should the APIs look like?
- Food for thought
Standard Telecommunications System Architecture

- Three separate layers (planes)
  - Data (Forwarding) Plane
  - Control Plane
  - Management (Configuration) Plane

- What constitutes a “plane”
  - The hardware
  - The algorithm
  - The interface

- It is the data plane that ultimately determines the system performance and functionality
P4 programs only define table structure

```
action send(PortId_t port) {
    ig_tm_md.unicast_egress_port = port;
}

action drop() {
    ig_dprsr_md.drop_ctl = 1;
}

action l3_switch(PortId_t port, bit<48> mac_da, bit<48> mac_sa) {
    hdr.ethernet.dst_addr = mac_da;
    hdr.ethernet.src_addr = mac_sa;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    send(port);
}

table ipv4_host {
    key = { hdr.ipv4.dst_addr : exact; }
    actions = { send; drop; l3_switch; }
    size = 131072;
}

table ipv4_lpm {
    key = { hdr.ipv4.dst_addr : lpm; }
    actions = { send; drop; l3_switch; }
    size = 12288;
    default_action = send(CPU_PORT);
}
```
Tables are populated by the Control Plane

A Table
Data Plane

Without the control plane populating the tables, data plane program is useless.
Definition

- **P4 Control Program Interface** is a set of methods that allow the **Control Plane** to manipulate or examine the state of
  - All Stateful Objects, defined in P4-programmable blocks
  - All Stateful Objects in the fixed-logic (non-P4) blocks
Stateful Objects

P4 Objects

- Tables
- Value Sets
- Externs
  - Counters
  - Meters
  - Registers
  - Action Profiles
  - Action Selectors
  - Hashes and Hash Algorithms
  - ...

Fixed Logic Blocks

- Ports
  - MAC
    - MAC counters
  - SerDes
- Traffic Manager
  - Memory Pools
  - Priority Groups
  - Queues
  - Schedulers
- Replication Engine
  - Multicast Groups

Anything that has a name in a P4 program can have an API
Problems to Solve

- **How should the APIs look like?**
  - Program-dependent and program-independent approaches
  - Defining the APIs for everything that is not a table

- **How to access the target?**
  - PCIe primer
  - Locally callable APIs
  - Remotely callable APIs (RPC)
Program-dependent and Program-independent APIs
Program-Dependent (Autogenerated APIs)

- Device is accessed via PCIe
- Low-level driver provides basic access
- “Fixed” APIs are manually coded
- The compiler autogenerates APIs for each P4 program
Automatically Generated Program-Dependent APIs (types)

```c
action send(PortId_t port) {
    ig_tm_md.unicast_egress_port = port;
}
action drop() {
    ig_dprsrm_md.drop_ctl = 1;
}
action l3_switch(PortId_t port, bit<48> mac_da, bit<48> mac_sa) {
    hdr.ethernet.dst_addr = mac_da;
    hdr.ethernet.src_addr = mac_sa;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    send(port);
}
table ipv4_host {
    key   = { hdr.ipv4.dst_addr : exact; }
    actions = { send; drop; l3_switch; }
    size   = 131072;
}
table ipv4_lpm {
    key   = { hdr.ipv4.dst_addr : lpm; }
    actions = { send; drop; l3_switch; }
    size   = 12288;
    default_action = send(CPU_PORT);
}
/* Representing action data for each action */
typedef struct p4_pd_myprog_send_action_spec {
    uint16_t port;
} p4_pd_myprog_send_action_spec_t;
typedef struct p4_pd_myprog_drop_action_spec {
} p4_pd_myprog_drop_action_spec_t;
typedef struct p4_pd_myprog_l3_switch_action_spec {
    uint16_t port;
    uint64_t mac_da;
    uint64_t mac_sa;
} p4_pd_myprog_l3_switch_action_spec_t;
/* Representing table keys for each table */
typedef struct p4_pd_myprog_ipv4_host_match_spec {
    uint32_t dst_addr;
} p4_pd_myprog_ipv4_host_match_spec_t;
typedef struct p4_pd_myprog_ipv4_lpm_match_spec {
    uint32_t dst_addr;
    uint32_t dst_addr_p_length;
} p4_pd_myprog_ipv4_lpm_match_spec_t;
```
/* Representing action data for each action */
typedef struct p4_pd_myprog_send_action_spec {
    uint16_t port;
} p4_pd_myprog_send_action_spec_t;

typedef struct p4_pd_myprog_drop_action_spec {
} p4_pd_myprog_drop_action_spec_t;

typedef struct p4_pd_myprog_l3_switch_action_spec {
    uint16_t port;
    uint64_t mac_da;
    uint64_t mac_sa;
} p4_pd_myprog_l3_switch_action_spec_t;

/* Representing table keys for each table */
typedef struct p4_pd_myprog_ipv4_host_match_spec {
    uint32_t dst_addr;
} p4_pd_myprog_ipv4_host_match_spec_t;

typedef struct p4_pd_myprog_ipv4_lpm_match_spec {
    uint32_t dst_addr;
    uint32_t dst_addr_p_length;
} p4_pd_myprog_ipv4_lpm_match_spec_t;

/* Table ipv4_host */
p4_pd_status_t p4_pd_myprog_ipv4_host_table_add_with_send(
    p4_pd_dev_target_t device_number,
    const p4_pd_myprog_ipv4_host_match_spec_t * key,
    const p4_pd_myprog_send_action_spec_t * data);

p4_pd_status_t p4_pd_myprog_ipv4_host_table_add_with_drop(
    p4_pd_dev_target_t device_number,
    const p4_pd_myprog_ipv4_host_match_spec_t * key);

p4_pd_status_t p4_pd_myprog_ipv4_host_table_add_with_l3_switch(
    p4_pd_dev_target_t device_number,
    const p4_pd_myprog_ipv4_host_match_spec_t * key,
    const p4_pd_myprog_l3_switch_action_spec_t * data);

/* Table ipv4_lpm */
p4_pd_status_t p4_pd_myprog_ipv4_lpm_table_add_with_send(
    p4_pd_dev_target_t device_number,
    const p4_pd_myprog_ipv4_lpm_match_spec_t * key,
    const p4_pd_myprog_send_action_spec_t * data);

p4_pd_status_t p4_pd_myprog_ipv4_lpm_table_add_with_drop(
    p4_pd_dev_target_t device_number,
    const p4_pd_myprog_ipv4_lpm_match_spec_t * key);

p4_pd_status_t p4_pd_myprog_ipv4_lpm_table_add_with_l3_switch(
    p4_pd_dev_target_t device_number,
    const p4_pd_myprog_ipv4_lpm_match_spec_t * key,
    const p4_pd_myprog_l3_switch_action_spec_t * data);
Adding and deleting a table entry

```c
#include "p4_pd.h"

p4_pd_myprog_ipv4_host_match_spec_t key;
p4_pd_myprog_send_action_spec_t data;

/* Prepare the key */
memset(&key, 0, sizeof(key));
key.dst_addr = 0xc0a80101;  // 192.168.1.1

/* Prepare the action data */
memset(&data, 0, sizeof(data));
data.port = 5;

/* Add an entry */
result =
p4_pd_myprog_ipv4_host_table_add_with_send(0, &key, &data);

/* Delete an entry */
result =
p4_pd_myprog_ipv4_host_table_delete(0, &key);
```

- Very easy to use, compact APIs
- A lot of compile-time checks
  - No way to specify a non-existing table
  - No way to specify a non-existing action
  - No way to specify wrong key or action data field
  - No way to specify incorrect action data for a given action
- Retrieving an entry might be challenging
  - We do not know what action data data structure to return
Program-Independent APIs

```c
#include "p4.h"

p4_key_t key; /* Abstract, opaque type */
p4_data_t data; /* Abstract, opaque type */

p4_key_init(key, "ipv4_host");
p4_key_field_set_exact(key, "dst_addr", 0xc0a80101);

p4_data_init(data, "ipv4_host", "send");
p4_data_field_set_exact(data, 5);

/* Adding an entry */
result = p4_table_entry_add(0, "ipv4_host", key, data);

/* Deleting an entry */
result = p4_table_entry_del(0, "ipv4_host", "port", key);

/* Retrieving an entry */
result = p4_table_entry_get(0, "ipv4_host", key, &data);

printf("Action: %s\n", p4_data_action_get(data));
for (int i = 0; i < p4_action_param_num_get(data); i++) {
    printf("%s: %d\n",
           p4_action_param_name_get(data, i),
           p4_action_param_value_get(data, i));
}
```
Program-Independent APIs (Analysis)

```c
#include "p4.h"

p4_key_t key;  /* Abstract, opaque type */
p4_data_t data;  /* Abstract, opaque type */

p4_key_init(key, "ipv4_host");
p4_key_field_set_exact(key, "dst_addr", 0xc0a80101);

p4_data_init(data, "ipv4_host", "send");
p4_data_field_set_exact(data, "port", 5);

/* Adding an entry */
result = p4_table_entry_add(0, "ipv4_host", key, data);

/* Deleting an entry */
result = p4_table_entry_del(0, "ipv4_host", key);

/* Retrieving an entry */
result = p4_table_entry_get(0, "ipv4_host", key, &data);

printf("Action: %s\n", p4_data_action_get(data));
for (int i = 0; i < p4_action_param_num_get(data); i++) {
    printf("%s: %d\n",
            p4_action_param_name_get(data, i),
            p4_action_param_value_get(data, i));
}
```

- Easy to use. More “verbose” APIs
- No compile time checks
  - Easy to specify a non-existing table/action/key/data field
- Not as efficient (string searches)
  - Fixed with assigned IDs
    - Also helps with typos
- Implementing generic algorithms becomes easy
- No need to have NxM functions
- No need to recompile control plane code
- Can be used to generate PD APIs
  - But not vice-versa
Program-Independent (Autogenerated Information) APIs

- Device is accessed via PCIe
- Low-level driver provides basic access
- “Fixed” APIs are manually coded
- APIs for P4 objects are generic
- The compiler autogenerates P4 object descriptions

Your Control Plane

Your High-Level (Semantic) APIs

"Fixed-function" APIs: PRE, TM, Ports, ...

P4 Object-Level APIs

P4 Object Definitions

P4 Compiler (bf-p4c)

Your Data Plane Program

Low-Level Driver

Device is accessed via PCIe

Low-level driver provides basic access

“Fixed” APIs are manually coded

APIs for P4 objects are generic

The compiler autogenerates P4 object descriptions
What about other objects?
Specialized Program-Dependent APIs

```c
Meter<bit<10>>(1024, MeterType_t.BYTES) acl_meter;

typedef struct p4_pd_meter_byte_spec {
    uint64_t cir_rate_bps;
    uint64_t cir_burst_kbits;
    uint64_t pir_rate_bps;
    uint64_t pir_burst_kbits;
} p4_pd_meter_byte_spec_t;

p4_pd_status_t p4_pd_myprog_meter_set_acl_meter(
    p4_pd_dev_target_t device_number,
    uint32_t meter_index,
    p4_pd_meter_byte_spec_t meter_byte_spec);

p4_pd_status_t p4_pd_myprog_meter_get_acl_meter(
    p4_pd_dev_target_t device_number,
    uint32_t meter_index,
    p4_pd_meter_byte_spec_t *meter_byte_spec);
```
Counter<bit<64>, bit<12>>(4096, CounterType_t::PACKETS_AND_BYTES) ipv4_stats;

typedef struct p4_pd_counter_value {
    uint64_t packets;
    uint64_t bytes;
} p4_pd_counter_value_t;

p4_pd_status_t p4_pd_myprog_counter_set_ipv4_stats (p4_pd_dev_target_t device_number, uint32_t counter_index, p4_pd_counter_value_t counter_value);

p4_pd_status_t p4_pd_myprog_counter_get_ipv4_stats (p4_pd_dev_target_t device_number, uint32_t counter_index, p4_pd_counter_value_t * counter_value);

p4_pd_status_t p4_pd_myprog_counter_set_range_ipv4_stats (p4_pd_dev_target_t device_number, uint32_t counter_index, uint32_t num_entries, p4_pd_counter_value_t counter_value);

p4_pd_status_t p4_pd_myprog_counter_get_range_ipv4_stats (p4_pd_dev_target_t device_number, uint32_t counter_index, uint32_t num_entries, p4_pd_counter_value_t * counter_value);

p4_pd_status_t p4_pd_myprog_counter_clear_ipv4_stats (p4_pd_dev_target_t device_number);

p4_pd_status_t p4_pd_myprog_counter_sync_ipv4_stats (p4_pd_dev_target_t device_number);
Logical Representation of Match-Action Tables

```c
action send(PortId_t port) {
    ig_tm_md.unicast_egress_port = port;
}

action drop() {
    ig_dprsr_md.drop_ctl = 1;
}

action l3_switch(PortId_t port, bit<48> mac_da, bit<48> mac_sa) {
    hdr.ethernet.dst_addr = mac_da;
    hdr.ethernet.src_addr = mac_sa;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    send(port);
}

table ipv4_host {
    key = { hdr.ipv4.dst_addr : exact; }
    actions = { send; drop; l3_switch; }
    size = 131072;
}

table ipv4_lpm {
    key = { hdr.ipv4.dst_addr : lpm; }
    actions = { send; drop; l3_switch; }
    size = 12288;
    default_action = send(CPU_PORT);
}
```

### Table: ipv4_host
- **Key**: hdr.ipv4.dst_addr
- **Actions**: send, drop, l3_switch
- **Size**: 131072

### Table: ipv4_lpm
- **Key**: hdr.ipv4.dst_addr
- **Actions**: send, drop, l3_switch
- **Size**: 12288

**Default Action**: send(CPU_PORT)
Representing Specialized Objects (Indirect Externs)

Counter<bit<64>, bit<12>>(4096, CounterType_t.PACKETS_AND_BYTES) ipv4_stats;
Meter<bit<10>>(ACL_METER_SIZE, MeterType_t.BYTES) acl_meter;

- The index is the primary key
  - Like an exact match table
- Action might be optional
- All entries might already in the table
  - No add/delete

<table>
<thead>
<tr>
<th>Key</th>
<th>(Volatile) Entry Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$COUNTER_INDEX</td>
<td>$COUNTER_SPEC_BYTES</td>
</tr>
</tbody>
</table>

The same approach can be used to represent any configuration!

<table>
<thead>
<tr>
<th>Key</th>
<th>Entry Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$METER_INDEX</td>
<td>$METER_SPEC_CIR_KBPS</td>
</tr>
<tr>
<td></td>
<td>$METER_SPEC_CIR_KBITS</td>
</tr>
<tr>
<td></td>
<td>$METER_SPEC_PIR_KBPS</td>
</tr>
<tr>
<td></td>
<td>$METER_SPEC_PIR_KBITS</td>
</tr>
</tbody>
</table>

That's exactly the idea behind Barefoot Runtime Interface (BRI)
Representing Fixed-Function Components

- **Fixed-function components are represented as tables too**

### port.port

<table>
<thead>
<tr>
<th>Key</th>
<th>Entry Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DEV_PORT</td>
<td>$SPEED=</td>
</tr>
<tr>
<td>$FEC=</td>
<td>$N_LANES=</td>
</tr>
<tr>
<td>$ENABLE=</td>
<td>$AUTONEG=</td>
</tr>
<tr>
<td>$TX_MTU=</td>
<td>$RX_MTU</td>
</tr>
<tr>
<td>$PORT_UP</td>
<td></td>
</tr>
</tbody>
</table>

### mirror.cfg

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
<th>Entry Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sid</td>
<td>normal</td>
<td>$session_enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$direction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$ucast_egress_port</td>
</tr>
<tr>
<td>$sid</td>
<td>coalescing</td>
<td>$session_enable</td>
</tr>
</tbody>
</table>

This field is volatile

### tf1.tm.queue.sched.cfg

<table>
<thead>
<tr>
<th>Key</th>
<th>Entry Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_id pg_queue</td>
<td>min_priority min_rate_enable dwrr_weight max_priority max_rate_enable scheduling_enable</td>
</tr>
</tbody>
</table>
Ascribing an API to an arbitrary extern

```c
enum e1_t { e1_value1, e1_value2 }
enum e2_t { e2_value1, e2_value2, e2_value3 }

extern ext<S> {
    ext(bit<32> param1, e1_t param2);
    e2_t method1(in S param1, in e2_t param2);
    e2_t method1(in S param1);
}

enum MeterType_t { PACKETS, BYTES }
enum MeterColor_t { RED, GREEN, YELLOW }

extern Meter<S> {
    Meter(bit<32> n_meters, MeterType_t type);
    MeterColor_t execute(in S index, in MeterColor_t color);
    MeterColor_t execute(in S index);
}
```

- What is the table format for the extern “ext”?
- Oh, that’s a Meter!

We can not currently derive the APIs or table formats based on P4 code alone.

We need to invent a new specialized (sub)language to go with P4!
Building a full API stack
PCI Express – the network inside your computer

<table>
<thead>
<tr>
<th>Version</th>
<th>Line code</th>
<th>Transfer rate per lane</th>
<th>Throughput (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>x1</td>
<td>x4</td>
</tr>
<tr>
<td>1.0</td>
<td>8b/10b</td>
<td>2.5 GT/s</td>
<td>0.250</td>
</tr>
<tr>
<td>2.0</td>
<td></td>
<td>5.0 GT/s</td>
<td>0.500</td>
</tr>
<tr>
<td>3.0</td>
<td>128b/130b</td>
<td>8.0 GT/s</td>
<td>0.985</td>
</tr>
<tr>
<td>4.0</td>
<td></td>
<td>16.0 GT/s</td>
<td>1.969</td>
</tr>
<tr>
<td>5.0</td>
<td></td>
<td>32.0 GT/s</td>
<td>3.938</td>
</tr>
</tbody>
</table>

Information in core section of TLP comes from Software Layer / Device Core.

Created by Transaction Layer

Appended by Data Link Layer

Appended by Physical Layer

© P4ica, LLC 2023
PCI Express – the network inside your computer

- Both memory and device registers are accessed using the same address space
- Pointers hold the addresses
- Based on the address the packet is routed to the proper place

```c
#include <stdint.h>

typedef uint64_t my_reg_t;
typedef volatile my_reg_t * my_reg_addr_t;

void my_reg_write(my_reg_addr_t addr, my_reg_t val)
{
    *addr = val; // movl %esi, (%rdi)
}

my_reg_t my_reg_read(my_reg_addr_t addr)
{
    return *addr; // movl (%rdi), %eax
}
```
Core Components (Tofino). Single Process

Your P4 Program

- bf-p4c
- bfrt.json
- context.json
- tofino.bin

Legend:
- Precompiled Binaries
- Provided Libraries
- Your code
- Generated results
- Dynamically linked binary
- Linux kernel Module

bfrt_python
BfRt API (C)

BfRt API (C++)

PipeMgr (C)
Fixed APIs (C)

Low-Level Driver

Target-syslibs
Target-utils

bf_switchd

Your Control-plane Program (Apps)

Debug CLI

bfshell
tcp port 9999

Linux User Space

Linux Kernel

tofino.bin

Legend:

- Precompiled Binaries
- Provided Libraries
- Your code
- Generated results
- Dynamically linked binary
- Linux kernel Module

© P4ica, LLC 2023
Full Stack (Tofino ASIC). Remote Program Load

Legend:
- Red: Precompiled Binaries
- Purple: Provided Libraries
- Green: Your code
- Blue: Generated results
- Cyan: Dynamically linked binary
- Orange: Linux kernel Module

Your P4 Program

BF Runtime Client
- Protobuf
- gRPC

BF Runtime Server
- BfRt API (C++)
- PipeMgr (C)
- Fixed APIs (C)

Low-Level Driver
- target-syslibs
- target-utils
- bf_switchd

BF Runtime Server
- bfrt.json
- context.json

Debug CLI
- tcp port 9999

Linux User Space
- bfshell

Legend:
- Red: Precompiled Binaries
- Purple: Provided Libraries
- Green: Your code
- Blue: Generated results
- Cyan: Dynamically linked binary
- Orange: Linux kernel Module

© P4ica, LLC 2023
What Have We Learned?

- P4 is just a part of the solution
- Control plane APIs are essential part of the data plane
- There are many ways to define control plane APIs
  - Program-dependent vs. program-independent
  - Specialized vs. generic (table-like)
  - Locally and remotely callable
- We still need to design a methodology for true automatic API generation for P4
Useful Materials (Click on the Links)

- P4ica
  - [Website](#) (under construction)
  - [Support Portal](#) (Course Materials and Recordings)
  - [Eventbrite](#) (Course signups)

- P4.org
  - [Website](#)  [Forum](#)  [Github](#)  [YouTube](#)

- Intel Connectivity Research Program
  - [Website](#)  [Forum](#)  [P4 Paper Collection](#) (no longer updated)

- Selected public videos of Vladimir, teaching P4

Q & A
Thank You!
intel® connectivity academy
by P4ica