Modeling hardware blocks of network ASICs using P4

Tourrilhes, Jean (*standing in for* Hardik Soni)

Introduction

Jean Tourrilhes is a researcher in the Network and Distributed Systems Laboratory, part of Hewlett Packard Labs. In a former life, Jean contributed to the OpenFlow specification. Jean is currently interested in congestion management and network virtualization.

Jean Tourrilhes is presenting this work on behalf of his colleagues, Arthur Simon, Hardik Soni, Khaled Diab and Puneet Sharma, also in the Network and Distributed Systems Lab (NDSL) of HPE.

Company Overview

HPE is the global edge-to-cloud company built to transform your business. How? By helping you connect, protect, analyze, and act on all your data and applications wherever they live, from edge to cloud, so you can turn insights into outcomes at the speed required to thrive in today’s complex world.

Functional simulator for network ASICs

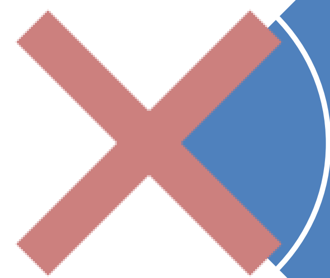
- Our use case :
  - Functional model of packet-processing features in network ASICs
  - Specifying fixed-function parsers and tables with precise implementation details
  - Accurate description of building-blocks like TCAMs, Hash, RAMs with P4 Tables
  - Leverage BMv2 to build a functional simulator for the ASIC

A specification that can compile

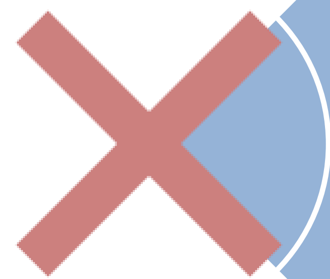
- Using P4 language to specify hardware features
  - Individual features of fixed-function network ASICs
  - More formalism and semantics than plain text, pseudo-code, C++ or system C
  - Easier to share and read amongst network designers
- Spec should be able to compile and provide a functional simulator
  - Remove effort duplication
  - Avoid spec diverging from model
  - Future: auto-generate spec from model

- Use P4-based specification
  - To document features
  - To iterate over functional design and evolve
  - To simulate the functionality and interaction among HW blocks

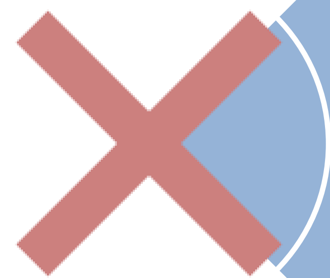
Challenges



Build functional simulator from feature specification

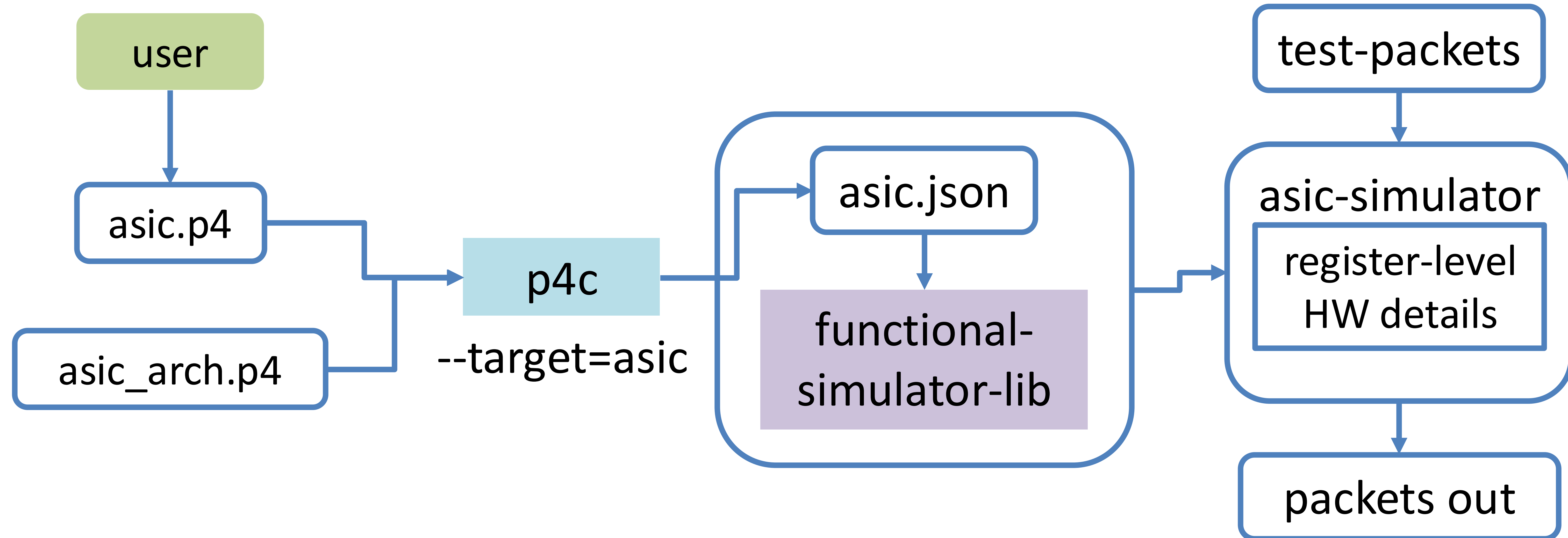


Specifying actual hardware implementation of fixed-function parsers



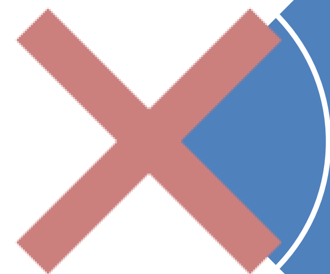
Expressing features of hardware matching blocks (TCAMs) with P4 Tables

Functional simulator for network ASICs



Challenges

Build functional simulator from feature specification

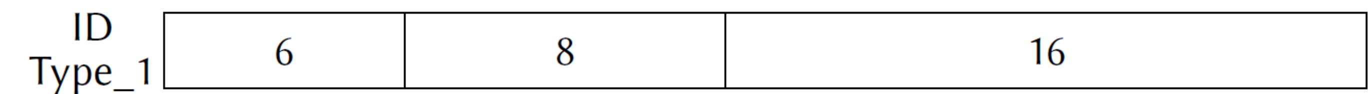


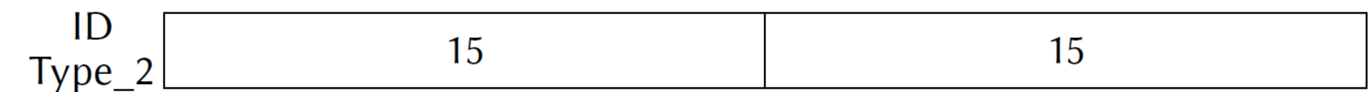
Specifying actual hardware implementation of fixed-function parsers

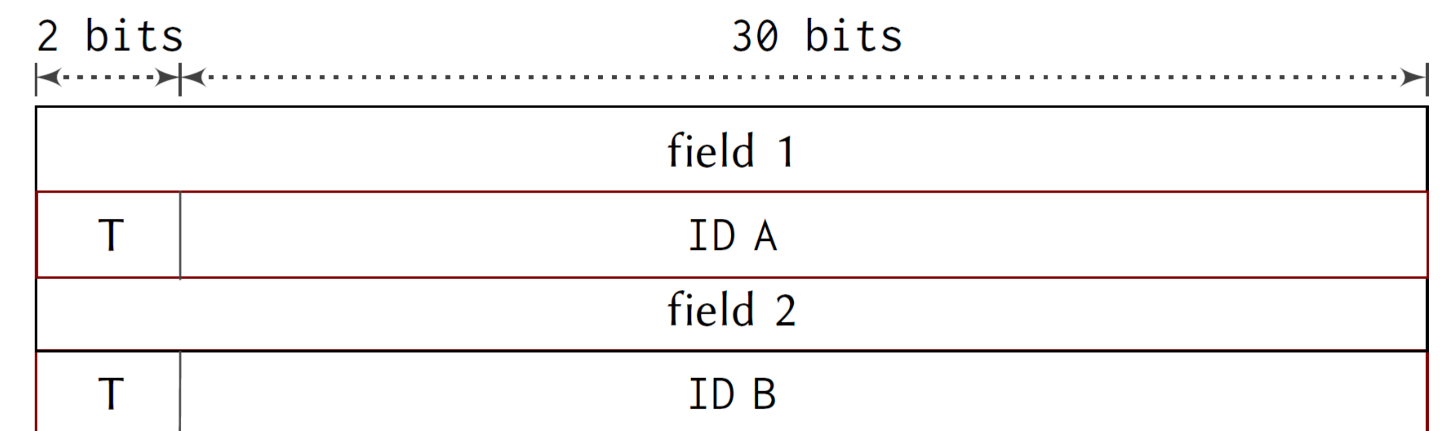
Expressing features of hardware matching blocks (TCAMs) with P4 Tables

Parsers: Specifying Implementation Detail

- Overloaded Fields in Protocol Headers
  - Hardware optimization for metadata and custom network stack
  - Different types of IDs (Type_1 and Type_2) in same header field
  - Example : port number or multicast group
- Limit feature modelling
  - Can’t explore handling of ambiguity in model
  - Can’t use actual test vectors from hardware6
- Conflict with P4 Type Nesting Rule
  - Must map each type to separate header field
  - P4 **header** can not have a member with **header_union** type







- ID fields with value A and B.
- Each ID can be of either Type_1 or Type_2

Parsers: Union using standard P4

ID Types

```
header id_1_h {
  bit<2>        type;
  bit<6>        if1;
  bit<8>        if2;
  bit<16>       if3;
}
header id_2_h {
  bit<2>        type;
  bit<15>       if1;
  bit<15>       if2;
}
header_union id_union_t {
  id_1_h        id_1;
  id_2_h        id_2;
}
header main_part_1_h {
  bit<32>       f1;
}
header main_part_2_h {
  bit<32>       f2;
}
struct headers_t {
  ... // other headers
  main_part_1_h   mh1;
  id_union_t      ida;
  main_part_2_h   mh2;
  id_union_t      idb;
}
```

Main header

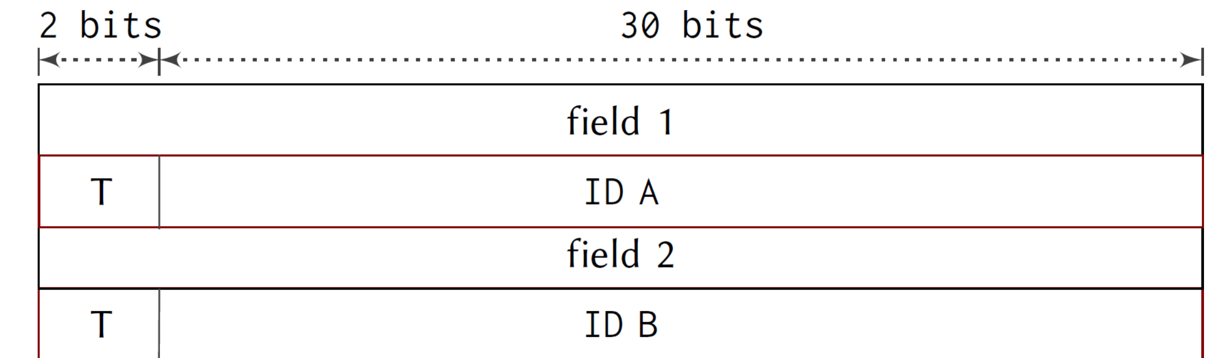
Parsing Union

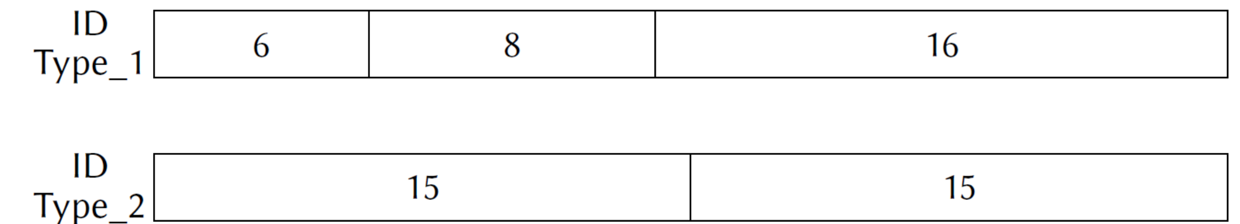
```
parser id_parser(packet_in b, out id_union_t id) {
  state start {
    transition select(b.lookahead<bit<2>>()) {
      (2w0): parse_type1;
      (2w1): parse_type2;
    }
  }
  state parse_type1 {
    b.extract(id.id_1);
    transition accept;
  }
  state parse_type2 {
    b.extract(id.id_2);
    transition accept;
  }
}
```

Parsing Main header

```
parser p(packet_in b, out headers_t h...) {
  state start {
    b.extract(h.mh1);
    id_parser.apply(b, h.ida);
    b.extract(h.mh2);
    id_parser.apply(b, h.idb);
  }
}
```

Main header





Parser: Union within header type

```
header id_1_h {
  bit<6>        if1;
  bit<8>        if2;
  bit<16>       if3;
}
header id_2_h {
  bit<15>       if1;
  bit<15>       if2;
}
header_union id_t {
  id_1_t        id_t_1;
  id_2_t        id_t_2;
}
struct type_id_t {
  bit<2>        type;
  id_t          id;
}
header main_h {
  bit<32>       f1;
  type_id_t     id_a;
  bit<32>       f2;
  type_id_t     id_b;
}
struct headers_t {
  ... // other headers
  main_h        mh;
}
```

ID Types

Main header

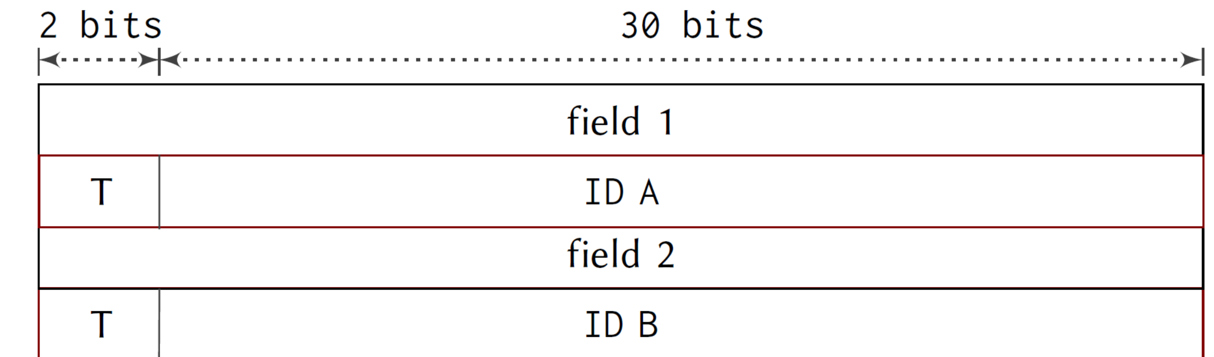
Parsing Main header

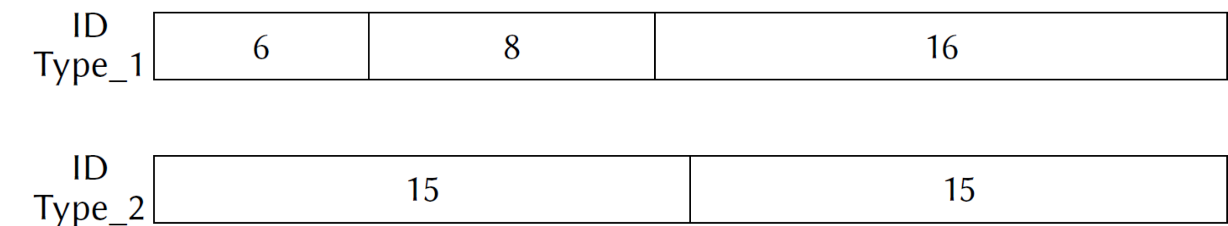
```
parser p(packet_in b, out headers_t h...) {
  state start {
    b.extract(h.mh);
    id_parser(h.mh.id_a.type, h.mh.id_a.id);
    id_parser(h.mh.id_b.type, h.mh.id_b.id);
  }
}
```

Parsing Union

```
parser id_parser(bit<2> type, header_union id) {
  state start {
    b.extract(h.mh);
    transition select(type) {
      (2w0): parse_type1;
      (2w1): parse_type2;
    }
  }
  state parse_type1 {
    id.id_t_1.setValid();
    transition accept;
  }
  state parse_type2 {
    id.id_t_2.setValid();
    transition accept;
  }
}
```

Main header





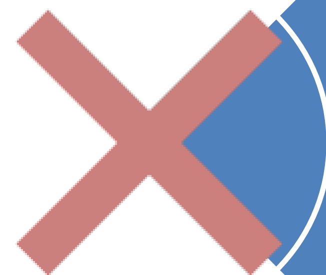
Parser: Relaxed Type Nesting Rules

- What about setValid, setInvalid and emit operations in case of type nesting?
  - For emit:
    - *A header is valid only if all of its members are valid.*
  - setInvalid()
    - *Invalidate all its members, recursively.*
  - setValid()
    - *operation on a header requires all its members to be valid.*
    - All the member headers and header_union should be explicitly set to valid before their containers are set to valid.
  - Compilers can make these checks at compile-time.
- Overall, minor change to semantic of language

Challenges

Build functional simulator from feature specification

Specifying actual hardware implementation of fixed-function parsers



Expressing features of hardware matching blocks (TCAMs) with P4 Tables

Modeling TCAM with P4 tables

- TCAM actual semantic different from P4 table
  - Order matters – rules have an **index**
  - For example: placement optimization
- Changes to control plane API
  - Add, Modify, Delete at a given location in tables
  - Augment APIs with **index** (entry_handle_t) as an *in* parameter (currently *out* parameter)
- Changes to matching API
  - Match Operation provides index on a successful hit
  - P4 table's compiler synthesized struct is augmented with an additional member bit<N> **index**.

An example change in add_entry API

Existing:

```
mt_add_entry(..., entry_handle_t& )
```

Added:

```
mt_add_entry(..., entry_handle_t )
```

```
struct apply_result(T, N) {
   bool hit;
   bool miss;
   action_list(T) action_run;
   bit<N> index;
}
```

Challenges

Build functional simulator from feature specification

Specifying actual hardware implementation of fixed-function parsers

Expressing features of hardware matching blocks (TCAMs) with P4 Tables

Conclusion

- Using P4 as a low-level language to model hardware blocks
  - P4 is a good language to specify features of fixed function network ASICs
  - Remove duplication : a specification that can compile
  - P4 language goal : from abstractions to specialization, low level descriptions

- P4 toolchain can be leveraged to simulate hardware blocks with detailed implementation
- Changes to P4 language
  - Flexible semantics for nested types in parser (unions)
  - Expressing TCAM semantic in table (index)

- Future: Expressing Hash Tables and RAM lookup

Thank You