v1.1rc1 Review

Leo Alterman // Barefoot Networks

September 14th, 2015

Blackboxes

- Vendor or standard library defines new types of P4 object using 'blackbox_type' syntax:

```
blackbox_type type_name {
      attribute attribute_name {
            type: int | string | block | expression | … ;
            [optional;]
      }
      method method_name (param_type param_name, … );
      // …
}
```

- Documentation provided alongside interface describes further restrictions imposed by compiler
  - Eg, a string attribute must be either 'packets' or 'bytes'

Whiteboxes

- Want to allow blocks of code to be reused and parameterized

- Group blocks of P4 code into 'whiteboxes'
  - They are **instantiated** the same way header instances are
  - They have **signatures** whose parameters are **bound at instantiation**
  - Access objects inside of module instances with traditional **dot notation**

```
// Declare
whitebox_type module_name (param_type param_name, … ) {
      // normal P4 code
}

// Instantiate
whitebox module_name instance_name (args);
```

TLV Header Processing

- Proposed solutions:
  - parser_counter blackbox
  - *new idea:* allow arbitrary expressions in set_metadata() source argument

- See example code

Deparsing

- Packet blackbox:
  - extract() method, replaces built-in parser extract() function
  - emit() method, added for deparsing
    - takes a header instance - outputs to header wire if it's valid, does nothing if not valid

- Packet blackbox instance is passed into parser and deparser whiteboxes as an argument (user never creates their own packet objects)

- Target architecture may or may not expose a deparser whitebox, may make it optional, etc.

Typed action signatures

- Require types on action and whitebox parameters
  - Parameters have directionality
    - **in**: May not be modified, can be a constant or other non-L-value
    - **out:** May be modified, implied to be uninitialized
    - **inout:** May be modified, already initialized with data
  - Use syntax similar to header field declaration

```
action foo (inout header ipv4_t ipv4, in bit<32> new_dst_addr)
{
      modify_field(ipv4.dstAddr, new_dst_addr);
}
```

Sequential action semantics

- Proposed approaches:
  - infer parallelism where possible
  - allow parallel actions through separate keyword

Supplemental Slides

Parser modularization

- Allow parse states to be called like subroutines
  - Use similar syntax
- Parsing now finishes by returning special 'accept' or 'reject' states
  - 'accept' returns to the caller (or exits the parser)
  - 'reject' throws a parser exception

```
// Declaration:
whitebox_type IPv4_Parser_t (out header ipv4_t myIpv4) {
      parser parse_ipv4 {
            extract(myIpv4);
            // imagine lots of sanity
            // checks here
            return accept;
      }
}
```

```
// Use (assume we always get IPv4 packets)
header ipv4_t global_ipv4;
whitebox IPv4_Parser_t ipv4_parser (global_ipv4);

parser my_parse_state {
      extract(ethernet);
      ipv4_parser.parse_ipv4();
      return select(global_ipv4.proto) {
          // ...
      }
}
```

Packet Blackbox Parser Example

```
whitebox_type parser_module (blackbox packet_t p, struct my_headers_t h) {

      parser start {
            p.extract(h.ethernet);
            return select (h.ethernet.ethertype) {
                  ETH_IPV4: parse_ipv4;
                  ETH_IPV6: parse_ipv6;
            }
      }

      parser ipv4 {
            p.extract(h.ipv4);
            return select (h.ipv4.proto) {
                  // …
            }
      }

      // …
}
```

Packet Blackbox Deparser Example

```
whitebox_type deparser_module (blackbox packet_t p, struct my_headers_t h) {

      // Architecture disallows anything inside this control func other
      // than packet 'emit' calls and checksum rewrites

      blackbox checksum ipv4_chksum {
            fields {
                  // list every h.ipv4 field here other than 'chksum'
            }
            algorithm : csum16;
      }

      control deparse {
            p.emit(h.fabric_header);
            p.emit(h.ethernet);
            ipv4_chksum.get_value(h.ipv4.chksum);
            p.emit(h.ipv4);
            p.emit(h.ipv6);
            // …
      }
}
```

Language/Architecture Separation

- Example (this is *all* architecture-supplied P4 code)
- Use type variables ("< ... >" syntax) to indicate pieces supplied by the user

```
whitebox_type parser <U> (
      out U user_data
);

whitebox_type ingress <U> (
      inout U user_data,
      inout metadata CONTROLS_INGRESS control
);

whitebox_type egress <U> (
      inout U user_data,
      inout metadata CONTROLS_EGRESS control
);
```

```
header_type CONTROLS_INGRESS_t {
      fields {
            bit<16> in_port;  /* input */
            bit<16> out_port; /* output */
            bit drop;         /* output */
      }
}
header_type CONTROLS_EGRESS_t {
      fields {
            bit<16> out_port; /* input */
            bit drop;         /* output */
      }
}
```