P4 2022 Workshop
May 24-26th

Dynamic P4 pipeline configuration

Anjali Singhai Jain, Intel
Hariharan Thantry, Google

Why do we need this?

- Supporting Infrastructure use cases
  - Blackholing (VM migration)
  - Supporting machine profiles (cloud, baremetal, native)
- Accelerating applications
  - Microservices, L4-L7 classification
  - Application Dedicated Queues (ADQ)

The basic idea….

- Underlying architecture has physical tables with fixed key width, and mechanisms to overlay logical P4 tables on this physical table.
  - Key construction: Selecting a set of fixed width elements that can populate the “key space” for the physical table.
  - Action selection: Initial proposal seeks to choose from an available set of action functions that are associated with the physical table.
  - Preconditions allow for adding this table to the lookup pipeline.

Key Point: The compiler still gets all the info upfront to decide the resource fitting for a given Target.

Proposal – the basic idea

- Exposes the physical table that cannot be used directly in the P4 pipeline but serves as a template.

```
@template_table
table flow_table {
  attributes {
    unit: 16 (in bits)
    width: 64 (in units)
    entries: 32 (number of logical tables)
    size: 16384 (total size across all tables)
  };
  actions {
   ....regular P4 actions....
  }
  selector {
     packet_field, (list of packet fields)
     metadata_field (list of metadata fields)
  }
};
```

- A template table is available to drive 1 of N lookups based on conditional evaluation of packet_field and metadata_field in the P4 source code

```
@template flow_table
{
  @expr_tag=0
  if (packet_field_A == val_A  | |packet_field_B ==
val_B) {
      ipv4.apply();
  }
  @expr_tag=3
  else if (meta_field_X == val_X) {
      l2.apply();
  }
};
```

Proposal – the basic idea (2)

- Insertion/removal of P4 tables into the pipeline at runtime through API
  - ***int p4_table_insert****(p4_table_name, template_tbl, conditional_expression, expression_tag, [packet_field or metadata_key with mask], [allowed-actions], default_action, [default_action_arg], size);*
    - p4_table_name: Canonical name to allow for future CRUD operations on the table.
    - template_tbl: Template table expressed in P4
    - conditional_expression: Conditional expression (machine friendly postfix?) composed from selector args only. Runtime validates.
    - expression_tag: Determines the order of evaluation (e.g. can take values 1, 2 to insert before l2_table)
    - [packet_field or metadata_key with mask]: List of packet/metadata fields with masks. Must be subset from the template table.
    - [allowed_actions]: List of allowed actions for this table. Must be subset from template table.
    - default_action: Default action to execute for miss.
    - [default_action_arg]: List of default action args (if necessary)
    - size: Size of this table.
  - ***int p4_table_remove****(p4_table_name)*
    - Removes a previously inserted table, erases all existing entries, and returns the resources to the system.

Blackholing use case

```
@template_table
table packet_type_group {
  attribute {
      unit: 8
      width: 1
      entries: 512
      size: 512 (each table is an entry each)
  }
  action {
     set_packet_type_grp;
     drop;
  }
  selector {
     local_metadata.ptype; // Value set in the parser.
  }
}
```

```
// P4 code
@expr_tag = 2
if (local_metadata.ptype == IPV4_NON_FRAG) {
   v4_grp_table.apply();
}
@expr_tag = 3
else if (local_metadata.ptype == IPV6_NON_FRAG) {
   v6_grp.apply()l
}
```

Blackholing use case - runtime

- We would like to drop all packets during VM migration. In this use case, we want to insert a new table as part of the same control flow. We can do this by inserting this new logical table into existing P4 profile.

assert(p4_table_insert**(blackhole_table, packet_type_group, [[local_meta.packet_type, [0x0, 0x0]*, ==]], 0, [local_meta.packet_type], [drop],drop, [], 1)* ***== 0)*;*

The above call would overlay an additional “blackhole_table” into the packet_type_group template that is the first in the sequence to be checked (expr_tag == 0), and is unconditionally executed (packet_type && 0x0 == 0x0). This table has only a single action drop, and hence will drop all packets.

After the control plane has migrated away all the VMs, the control plane can remove this table to resume normal operation.

***assert(p4_table_remove**(blackhole_table))*

** Contains (mask, value), to be executed if (local_meta.packet_type && mask == value && mask)*

Application match use case

```
@template_table
table ipv4_tcp{
  attribute {
      unit: 32
      width: 6
      entries: 2
      size: 1K (each table is 512 entry)
  }
  action {
     set_application_queue;
     drop;
  }
  selector {
     ip_src; ip_dest; port_rsc; port_dest; word_32; word_33;
  }
}
```

```
// P4 code
@expr_tag = 2
if (user_configdata == tuple_1_2_3) {
   v4_3tuple_table.apply();
}
@expr_tag = 3
else if (user_configdata == tuple_1_word_32) {
   v4_2tuple_table.apply()l
}
```

Application SW Config at Runtime

- Runtime API hooks from Application that go into the P4 Target Runtime Server that can define the match for an actual table from the template
  - Query the template table
  - Add table with the match specified.
  - Remove table

Result of this match may identify the application flows themselves and/or used to provide some user meta data hint to the application.

Notices and Disclaimers

- Intel technologies may require enabled hardware, software or service activation.
- No product or component can be absolutely secure.
- Your costs and results may vary.
-

Thank You