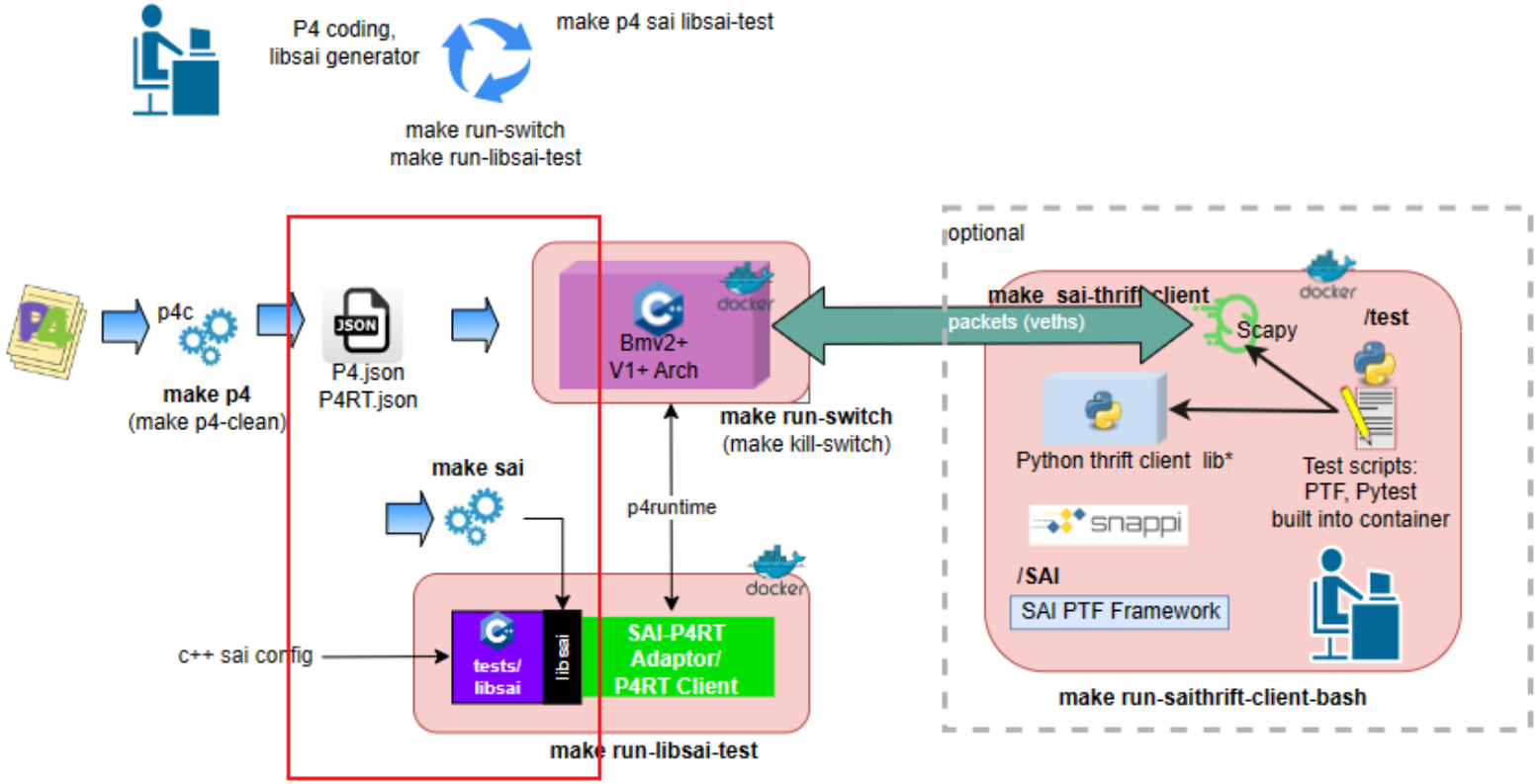
DASH SAI Generation Improvements

Riff Jiang



SAI generator

https://github.com/sonic-net/DASH/blob/main/dash-pipeline/SAI/sai_api_gen.py

Motivations

- Readability
  - Everything is dictionary-based.
  - Hard to follow the code, if you are not familiar with the P4RT data structure.

```
def extract_action_data(program):
    action_data = {}
    sai_enums = get_sai_enums(program)
    for action in program[ACTIONS_TAG]:
        preable = action[PREAMBLE_TAG]
        id = preable['id']
        name = preable[NAME_TAG].split('.')[-1]
        params = []
        if PARAMS_TAG in action:
            for p in action[PARAMS_TAG]:
                param = dict()
                param['id'] = p['id']
                param[NAME_TAG] = p[NAME_TAG]
                if STRUCTURED_ANNOTATIONS_TAG in p:
                    p4_annotation_to_sai_attr(p, param)
                else:
                    param['type'], param['field'] = get_sai_key_type(int(p[BITWIDTH_TAG]), p[NAME_TAG], p[NAME_TAG])
                    for sai_enum in sai_enums:
                        if param[NAME_TAG] == sai_enum['name']:
                            param['type'] = 'sai_' + param[NAME_TAG] + '_t'
                            param['field'] = 's32'
                            param['default'] = 'SAI_' + param[NAME_TAG].upper() + '_INVALID'
                param['bitwidth'] = p[BITWIDTH_TAG]
                params.append(param)
        action_data[id] = {'id': id, NAME_TAG: name, PARAMS_TAG: params}
    return action_data
```

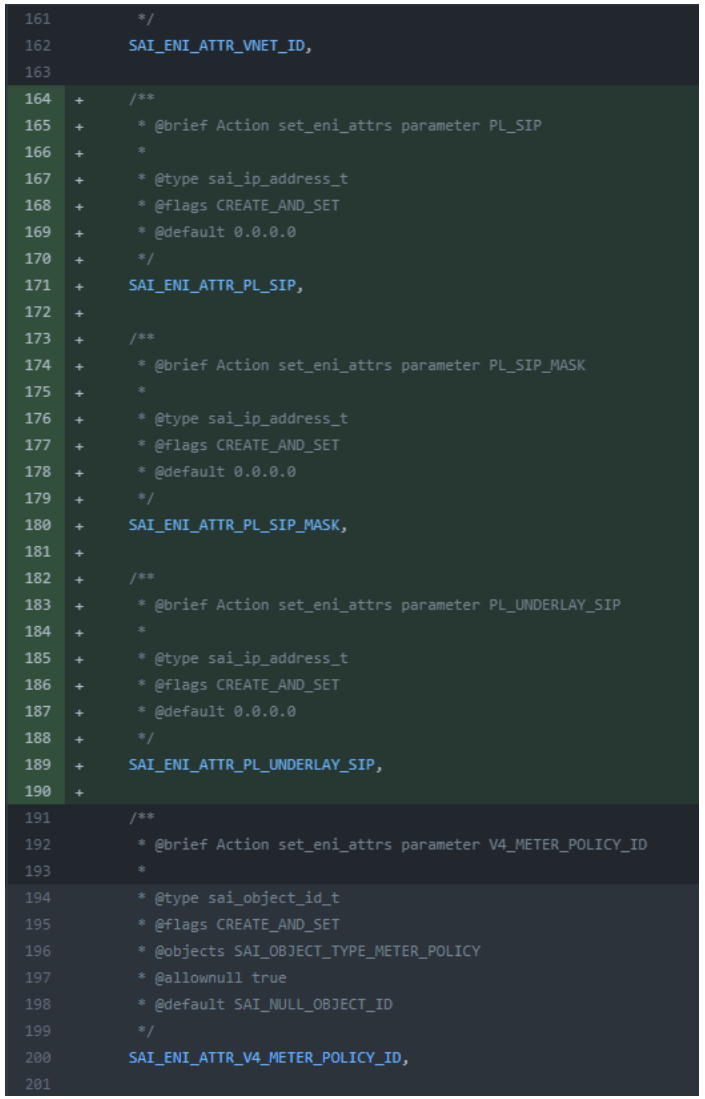
Motivations

- Being vaguely smart
  - Hard to understand and inconsistent @name annotation format
    - @name("outbound_routing|dash_outbound_routing")
    - @name("meta.dash_acl_group_id:dash_acl_group_id")
  - Type deduction using variable name and its parent field name
    - If the name contains “ip”, then it should be an ip. But some places, people use “addr”…
  - Fake code for SAI generation, such as metering bucket byte counters.

```
#ifdef TARGET_BMV2_V1MODEL
    counter(MAX_METER_BUCKETS, CounterType.bytes) meter_bucket_inbound;
    counter(MAX_METER_BUCKETS, CounterType.bytes) meter_bucket_outbound;
#endif // TARGET_BMV2_V1MODEL
    action meter_bucket_action(
            @SaiVal[type="sai_uint64_t", isreadonly="true"] bit<64> outbound_bytes_counter,
            @SaiVal[type="sai_uint64_t", isreadonly="true"] bit<64> inbound_bytes_counter,
            @SaiVal[type="sai_uint32_t", skipattr="true"] bit<32> meter_bucket_index) {
        // read only counters for SAI api generation only
        meta.meter_bucket_index = meter_bucket_index;
    }
```

Motivations

- Breaking change on updates
  - Any new table key or action parameter could lead attributes to be generated in the middle of the existing SAI attributes.
  - Merging parameters from multiple actions / merging table keys and actions parameters makes things even worse.
- Certain attributes cannot be generated correctly due to unsupported match in BMv2
  - Range match, list match



```
     */
    SAI_ENI_ATTR_VNET_ID,

+   /**
+    * @brief Action set_eni_attrs parameter PL_SIP
+    *
+    * @type sai_ip_address_t
+    * @flags CREATE_AND_SET
+    * @default 0.0.0.0
+    */
+   SAI_ENI_ATTR_PL_SIP,
+
+   /**
+    * @brief Action set_eni_attrs parameter PL_SIP_MASK
+    *
+    * @type sai_ip_address_t
+    * @flags CREATE_AND_SET
+    * @default 0.0.0.0
+    */
+   SAI_ENI_ATTR_PL_SIP_MASK,
+
+   /**
+    * @brief Action set_eni_attrs parameter PL_UNDERLAY_SIP
+    *
+    * @type sai_ip_address_t
+    * @flags CREATE_AND_SET
+    * @default 0.0.0.0
+    */
+   SAI_ENI_ATTR_PL_UNDERLAY_SIP,
+

    /**
     * @brief Action set_eni_attrs parameter V4_METER_POLICY_ID
     *
     * @type sai_object_id_t
     * @flags CREATE_AND_SET
     * @objects SAI_OBJECT_TYPE_METER_POLICY
     * @allownull true
     * @default SAI_NULL_OBJECT_ID
     */
    SAI_ENI_ATTR_V4_METER_POLICY_ID,
```

Goals

- Better readability.
- Explicit or smart with certainty.
- Tools to ensure ABI compatibility.
- Ensure generated code can match what we have in SAI.

Better readability

- Class instead of dictionaries.
- Comments to show what is parsed.
- Clear object hierarchy.
- Explicit type hinting.

```
# At high level, the hiredarchy of the SAI objects is as follows:
#
# DASHSAIExtensions                    : All DASH SAI extensions.
# |- SAIEnum                           : A single enum type.
# |  |- SAIEnumMember                  : A single enum member withi
# |- SAIAPISet                         : All information for a sing
#    |- SAIAPITableData                : All information for a sing
#       |- SAIAPITableKey              : Information of a single P4
#       |- SAIAPITableAction  <-------| : Information of a single P4
#          |- SAIAPITableActionParam -| : Information of a single P4
#
```

```python
@sai_parser_from_p4rt
class SAIAPITableKey(SAIAPITableAttribute):
    '''
    This class represents a single SAI API table key and provides parser from the P4Runtime table key object.
    '''
    def __init__(self):
        super().__init__()
        self.ip_is_v6_field_id: int = 0

    def parse_p4rt(self, p4rt_table_key: Dict[str, Any]) -> None:
        '''
        This method parses the P4Runtime table key object and populates the SAI API table key object.

        Example P4Runtime table key object:

            {
                "id": 1,
                "name": "meta.vnet_id:vnet_id",
                "bitwidth": 16,
                "matchType": "EXACT"
            },
            {
                "id": 2,
                "name": "hdr.ipv4.src_addr:sip",
                "bitwidth": 32,
                "matchType": "EXACT"
            }
        '''

        self.bitwidth = int(p4rt_table_key[BITWIDTH_TAG])
```

Explicit or smart with certainty

- Explicit annotation. Not more tricks on variable names.
  - https://github.com/sonic-net/DASH/blob/main/dash-pipeline/bmv2/README.md.
- Simplify type handling with type info and type solver (only uses variable size).
- Leverage not only P4RT, but also P4 IR.
  - Creates variable relationships from the code.

Tools to ensure ABI compatibility

- order annotation to ensure API functions and attributes can be generated in a specific order.

```
@SaiTable[name = "meter_policy", api = "dash_meter", order = 1, isobject="true"]
table meter_policy {
    key = {
        meta.meter_policy_id : exact;
    }
    actions = {
        check_ip_addr_family;
    }
}

action set_policy_meter_class(bit<16> meter_class) {
    meta.policy_meter_class = meter_class;
}

@SaiTable[name = "meter_rule", api = "dash_meter", order = 2, isobject="true"]
table meter_rule {
    key = {
        meta.meter_policy_id: exact @SaiVal[type="sai_object_id_t", isresourcetype="true", objects="METER_POLICY"];
        hdr.ipv4.dst_addr : ternary @SaiVal[name = "dip", type="sai_ip_address_t"];
    }

 actions = {
        set_policy_meter_class;
        @defaultonly NoAction;
    }
    const default_action = NoAction();
}
```

More...

- Feature supports
  - list match, range match in SAI header and libsai generation
  - Use counter to generate the stats attributes
- Clean up works

Future works

- Stats support
- Better leverage the counters and registers
- SAI events generation
- …