SONiC YANG MODEL GUIDELINES

Revision

| Rev | Date | Author | Change Description |
|:---:|:-----------:|:------------------:|-----------------------------------| | 1.0 | 22 Aug 2019 | Praveen Chaudhary | Initial version | | 1.0 | 11 Sep 2019 | Partha Dutta | Adding additional steps for SONiC YANG | | 1.1 | 15 Dec 2023 | Jingwen Xie | Added rules for List Keys |

References

| References | Date/Version | Link |
|:-----------------------:|:------------------:|:----------------------------------:| | RFC 7950 | August 2016 | https://tools.ietf.org/html/rfc7950 | | Management Framework | 0.9 | https://github.com/sonic-net/SONiC/pull/436 |

Terminology and Acronyms

| Acronyms | Description/Expansion | |:-------------------------:|:--------------------:| | ABNF | Augmented Backus-Naur Form | | XPath | XML Path Language | | CVL | Configuration Validation Library |

Overview

This document lists the guidelines, which will be used to write YANG Modules for SONiC. These YANG Modules (called SONiC YANG models) will be primarily based on or represent the ABNF.json of SONiC, and the syntax of YANG models must follow RFC 7950 (https://tools.ietf.org/html/rfc7950). Details of config in format of ABNF.json can be found at https://github.com/sonic-net/SONiC/wiki/Configuration.

These YANG models will be used to verify the configuration for SONiC switches, so a library which supports validation of configuration on SONiC Switch must use these YANG Models. List of such Libraries are: 1.) Configuration Validation Library. (CVL). YANG models, which are written using these guidelines can also be used as User End YANG Models, i.e North Bound configuration tools or CLI can provide config data in sync with these YANG models. For example SONiC Management Framework uses SONiC YANG models as Northbound Management YANG and for configuration validation purpose also.

Guidelines

1. All schema definitions related to a feature should be written in a single YANG model file. YANG model file is named as 'sonic-{feature}.yang', e.g. for ACL feature sonic-acl.yang is the file name. It is best to categorize YANG Modules based on a networking components. For example, it is good to have separate modules for VLAN, ACL, PORT and IP-ADDRESSES etc.

```
sonic-acl.yang sonic-interface.yang sonic-port.yang sonic-vlan.yang
```

2. It is mandatory to define a top level YANG container named as 'sonic-{feature}' i.e. same as YANG model name. For example, sonic-acl.yang should have 'sonic-acl' as top level container. All other definition should be written inside the top level container.

Example :

YANG

```
yang module sonic-acl { container sonic-acl { ..... ..... } }
```

3. Define namespace as "http://github.com/sonic-net/{model-name}".

Example :

YANG

```
yang module sonic-acl { namespace "http://github.com/sonic-net/sonic-acl"; ..... ..... }
```

4. Use 'revision' to record revision history whenever YANG model is changed. Updating revision with appropriate details helps tracking the changes in the model.

Example :

YANG

```yang module sonic-acl { revision 2019-09-02 { description "Added must expression for ACL_RULE_LIST."; // Updated with new change details }
```

```
revision 2019-09-01 {
    description
        "Initial revision.";
}
.....
.....

} ```
```

5. Each primary section of ABNF.json (i.e a dictionary in ABNF.json) for example, VLAN, VLAN_MEMBER, INTERFACE in ABNF.json will be mapped to a container in YANG model.

Example: Table VLAN will translate to container VLAN.

ABNF

`yang "VLAN": { "Vlan100": { "vlanid": "100" } }` will translate to:

YANG

`-- yang container VLAN { //"VLAN" mapped to a container list VLAN_LIST { key name; leaf vlanid { type uint16; } } }`

6. Each leaf in YANG module should have same name (including their case) as corresponding key-fields in ABNF.json.

Example: Leaf names are same PACKET_ACTION, IP_TYPE and PRIORITY, which are defined in ABNF.

ABNF

`"NO-NSW-.....|....." : { "PACKET_ACTION": "FORWARD", "IP_TYPE": "IPv6ANY", "PRIORITY": "955520", ..... ..... },`

YANG

`yang leaf PACKET_ACTION { ..... } leaf IP_TYPE { ..... } leaf PRIORITY { ..... }`

7. Use IETF data types for leaf type first if applicable (RFC 6021) . Declare new type (say SONiC types) only if IETF type is not applicable. All SONiC types must be part of same header type or common YANG model.

Example:

YANG

```yang leaf SRC_IP { type inet:ipv4-prefix; <<<< }

```
leaf DST_IP {
    type inet:ipv4-prefix;
}
```

```

8. Data Node/Object Hierarchy of the an objects in YANG models will be same as for all the fields at same hierarchy in Config DB. If any exception is created then it must be recorded properly with comment under object level in YANG models. To see an example of a comment, please refer to the step with heading as "Comment all must, when and patterns conditions." below.

For Example:

"Family" of VLAN_INTERFACE and "IP_TYPE" of ACL_RULE should be at same level in YANG model too.

ABNF

```
"VLAN_INTERFACE": { "Vlan100|2a04:f547:45:6709::1/64": { "scope":
"global", "family": "IPv6" } } "ACL_RULE": { "NO-NSW-PACL-V4|
DEFAULT_DENY": { "PACKET_ACTION": "DROP", "IP_TYPE": "IPv4ANY",
"PRIORITY": "0" } }
```

YANG

In YANG, "Family" of VLAN_INTERFACE and "IP_TYPE" of ACL_RULE is at same level. ```yang container VLAN_INTERFACE { description "VLAN_INTERFACE part of config_db.json"; list VLAN_INTERFACE_LIST { .... leaf family { type sonic-head:ip-family; } } }

container ACL_RULE { description "ACL_RULE part of config_db.json"; list ACL_RULE_LIST { .... leaf IP_TYPE { type sonic-head:ip_type; } } } ```

9. If an object is part of primary-key in ABNF.json, then it should be a key in YANG model. In YANG models, a primary-key from ABNF.json can be represented either as name of a Container object or as a key field in List object. Exception must be recorded in YANG Model with a comment in object field. To see an example of a comment, please refer to the step with heading as "Comment all must, when and patterns conditions." below. Though, key names are not stored in Redis DB, use the same key name as defined in ABNF schema.

Example: VLAN_MEMBER dictionary in ABNF.json has both vlan-id and ifname part of the key. So YANG model should have the same keys.

ABNF

```
"VLAN_MEMBER": { "Vlan100|Ethernet0": { //<<<< KEYS
"tagging_mode": "untagged" } } ;Defines interfaces which are members of a vlan key = VLAN_MEMBER_TABLE:"Vlan"vlanid:ifname ;
```

YANG

```
yang container VLAN_MEMBER { description "VLAN_MEMBER part of config_db.json"; list ..... { key "vlan-name ifname";//<<<< KEYS } }
```

10. If any key used in current table refers to other table, use leafref type for the key leaf definition.

Example :

ABNF

```
key: ACL_RULE_TABLE:table_name:rule_name ..... .....
```

YANG

```yang ..... container ACL_TABLE { list ACL_TABLE_LIST { key table_name;

    leaf table_name {
        type string;
    }
    .....
    .....
}
```

} container ACL_RULE { list ACL_RULE_LIST { key "table_name rule_name";

```
    leaf table_name {
        type leafref {
            path "../../../ACL_TABLE/ACL_TABLE_LIST/aclname"; //Refers to
        }
    }

    leaf rule_name {
        type string;
    }
    .....
    .....
}

} ```
```

11. Mapping tables in Redis are defined using nested 'list'. Use 'sonic-ext:map-list "true";' to indicate that the 'list' is used for mapping table. The outer 'list' is used for multiple instances of mapping. The inner 'list' is used for mapping entries for each outer list instance.

Example :

ABNF

``` ; TC to queue map ;SAI mapping - qos_map with SAI_QOS_MAP_ATTR_TYPE == SAI_QOS_MAP_TC_TO_QUEUE. See saiqosmaps.h key = "TC_TO_QUEUE_MAP_TABLE:"name ;field tc_num = 1*DIGIT ;values queue = 1*DIGIT; queue index

```

YANG

```yang container TC_TO_QUEUE_MAP { list TC_TO_QUEUE_MAP_LIST { key "name"; sonic-ext:map-list "true"; //special annotation for map table

        leaf name {
            type string;
        }

        list TC_TO_QUEUE_MAP_LIST { //this is list inside list for storing
            key "tc_num qindex";
            leaf tc_num {
                type string {
                    pattern "[0-9]?";
                }
            }
            leaf qindex {
                type string {
                    pattern "[0-9]?";
```

```
                }
            }
        }
    }
}
```

12. 'ref_hash_key_reference' in ABNF schema is defined using 'leafref' to the referred table.

Example :

ABNF

``` ; QUEUE table. Defines port queue. ; SAI mapping - port queue.

key = "QUEUE_TABLE:"port_name":queue_index queue_index = 1*DIGIT
port_name = ifName queue_reference = ref_hash_key_reference ;field value
scheduler = ref_hash_key_reference; reference to scheduler key
wred_profile = ref_hash_key_reference; reference to wred profile key

```

YANG

```yang container sonic-queue { container QUEUE { list QUEUE_LIST { .....
leaf scheduler { type leafref { path "/sch:sonic-scheduler/sch:SCHEDULER/
sch:name"; //Reference to SCHEDULER table } }

        leaf wred_profile {
            type leafref {
                path "/wrd:sonic-wred-profile/wrd:WRED_PROFILE/wrd:name";
            }
        }
    }
}

} ```

## 13. To establish complex relationship and constraints among multiple tables use 'must' expression. Define appropriate error message for reporting to Northbound when condition is not met. For existing feature, code logic could be reference point for deriving 'must' expression.

Example:
```

YANG

```
yang must "(/sonic-ext:operation/sonic-ext:operation != 'DELETE')
or " + "count(../../ACL_TABLE[aclname=current()]/ports) =
0" { error-message "Ports are already bound to this rule."; }
```

14. Define appropriate 'error-app-tag' and 'error' messages for in 'length', 'pattern', 'range' and 'must' statement so that management application can use it for error reporting.

Example:

YANG

```yang module sonic-vlan { .... .... leaf vlanid { mandatory true; type uint16 { range "1..4095" { error-message "Vlan ID out of range"; error-app-tag vlanid-invalid; } } } .... .... }

```

15. All must, when, pattern and enumeration constraints can be derived from .h files or from code. If code has the possibility to have unknown behavior with some config, then we should put a constraint in YANG models objects. Also, Developer can put any additional constraint to stop invalid configuration. For new features, constraints may be derived based on low-level design document.

For Example: Enumeration of IP_TYPE comes for aclorch.h ```

define IP_TYPE_ANY "ANY"

define IP_TYPE_IP "IP"

define IP_TYPE_NON_IP "NON_IP"

define IP_TYPE_IPv4ANY "IPV4ANY"

define IP_TYPE_NON_IPv4 "NON_IPv4"

define IP_TYPE_IPv6ANY "IPV6ANY"

define IP_TYPE_NON_IPv6 "NON_IPv6"

define IP_TYPE_ARP "ARP"

define IP_TYPE_ARP_REQUEST "ARP_REQUEST"

define IP_TYPE_ARP_REPLY "ARP_REPLY"

``` Example of When Statement: Orchagent of SONiC will have unknown behavior if below config is entered, So YANG must have a constraint. Here SRC_IP is IPv4, where as IP_TYPE is IPv6.
```

ABNF:

``` "ACL_RULE": { "NO-NSW-PACL-V4|Rule_20": { "PACKET_ACTION": "FORWARD", "DST_IP": "10.186.72.0/26", "SRC_IP": "10.176.0.0/15", "PRIORITY": "999980", "IP_TYPE": "IPv6" },

```

YANG:

```
yang choice ip_prefix { case ip4_prefix { when "boolean(IP_TYPE[.='ANY' or .='IP' or .='IPV4' or .='IPV4ANY' or .='ARP'])"; leaf SRC_IP { type inet:ipv4-prefix; } leaf DST_IP { type inet:ipv4-prefix; } } case ip6_prefix { when "boolean(IP_TYPE[.='ANY' or .='IP' or .='IPV6' or .='IPV6ANY'])"; leaf SRC_IPV6 { type inet:ipv6-prefix; } leaf DST_IPV6 { type inet:ipv6-prefix; } } } Example of Pattern: If PORT Range should be "<0-65365> - <0-65365>" leaf L4_DST_PORT_RANGE { type string { pattern '([0-9]{1,4}|[0-5][0-9]{4}|[6][0-4][0-9]{3}|[6][5][0-2][0-9]{2}|[6][5][3][0-5]{2}|[6][5][3][6][0-5])-([0-9]{1,4}|[0-5][0-9]{4}|[6][0-4][0-9]{3}|[6][5][0-2][0-9]{2}|[6][5][3][0-5]{2}|[6][5][3][6][0-5])'; } }
```

16. Comment all must, when and patterns conditions. See example of comment below.

Example:

YANG

```
yang leaf family { /* family leaf needed for backward compatibility Both ip4 and ip6 address are string in IETF RFC 6020, so must statement can check based on : or ., family should be IPv4 or IPv6 according. */ must "(contains(../ip-prefix, ':') and current()='IPv6') or (contains(../ip-prefix, '.') and current()='IPv4')"; type sonic-head:ip-family; }
```

17. If a List object is needed in YANG model to bundle multiple entries from a Table in ABNF.json, but this LIST is not a valid entry in config data, then we must define such list as _LIST .

For Example: Below entries in PORTCHANNEL_INTERFACE Table must be part of List Object in YANG model, because variable number of entries may be present in config data. But there is no explicit list in config data. To support this, a list object with name PORTCHANNEL_INTERFACE_LIST should be added in YANG model.

ABNF:

```
"PORTCHANNEL_INTERFACE": { "PortChannel01|10.0.0.56/31": {},
"PortChannel01|FC00::71/126": {}, "PortChannel02|10.0.0.58/31":
{}, "PortChannel02|FC00::75/126": {} ... }
```

YANG

```yang container PORTCHANNEL_INTERFACE {

    description "PORTCHANNEL_INTERFACE part of config_db.json";

    list PORTCHANNEL_INTERFACE_LIST {<<<<<
    .....
    .....
    }

} ```

## 18. In some cases it may be required to split an ABNF table into multiple YANG lists based on the data stored in the ABNF table. In this case it is crucial to ensure that the List keys are non-overlapping, unique, and unambiguous.

**Strategies for Ensuring Unique and Unambiguous Keys**: Utilize composite keys that have a different number of key elements to distinguish lists. Need to mention that different key names do not count as unambiguous model.

**ABNF**

```
"INTERFACE" : { "Ethernet1" : { "vrf-name": "vrf1" } "Ethernet1|
10.184.230.211/31": { } }
```

**Example 1: Key with different number of elements(composite keys - Allowed case)**

`INTERFACE` table stores VRF names to which an interface belongs, also it stores IP address of each interface. Hence it is needed to split them into two different YANG lists.

```yang ..... container INTERFACE { list INTERFACE_LIST { // 1st list key ifname;

    leaf ifname {
        type leafref {
        ......
        }
    }
    leaf vrf-name {
```
```

```
        type leafref {
        ......
        }
    }
    ......
}

list INTERFACE_IPADDR_LIST { //2nd list
    key "ifname ip_addr"

    leaf ifname {
        type leafref {
        ......
        }
    }
    leaf ip_addr {
          type inet:ipv4-prefix;
    }
    ......
}
```

} ..... ``` In the example above if the config DB contains an INTERFACE table with single key element then it will be associted with the INTERFACE_LIST and if contains 2 key elements then it will be associated with INTERFACE_IPADDR_LIST

Example 2: Keys with same number of elements of same type (NOT Allowed case 1)

```yang ..... container NOT_SUPPORTED_INTERFACE { list NOT_SUPPORTED_INTERFACE_LIST { // 1st list key ifname; leaf ifname { type string; } // ... }

```
list NOT_SUPPORTED_INTERFACE_ANOTHER_LIST { // Negative case
    key ifname;
    leaf ifname {
        type string;
    }
    // ...
}
```

} ..... ```

In the example above if the config DB contains an NOT_SUPPORTED_INTERFACE table with key Ethernet1 then it would match with both the list, this is an overlapping scenario
```

Example 3: Keys with same number of elements of same type (NOT Allowed case 2)

```yang ..... container NOT_SUPPORTED_TELEMETRY_CLIENT { list NOT_SUPPORTED_TELEMETRY_CLIENT_DS_LIST { // 1st list key "prefix name";

    leaf prefix {
        type string {
            pattern "DestinationGroup_" + ".*";
        }
    }

    leaf name {
        type string;
    }

    leaf dst_addr {
        type ipv4-port;
    }
}

list NOT_SUPPORTED_TELEMETRY_CLIENT_SUB_LIST { // Negative case
    key "prefix name";

    leaf prefix {
        type string {
            pattern "Subscription_" + ".*";
        }
    }

    leaf name {
        type string;
    }

    leaf dst_group {
        must "(contains(../../TELEMETRY_CLIENT_DS_LIST/prefix, current()))
        type string;
    }
}

} ..... ``` In the example above if the config DB contains an NOT_SUPPORTED_TELEMETRY_CLIENT table with key "DestinationGroup|HS", then it would correspond to the NOT_SUPPORTED_TELEMETRY_CLIENT_DS_LIST and NOT_SUPPORTED_TELEMETRY_CLIENT_SUB_LIST, this is an overlapping scenario
```

Example 4: keys with same number of elements and different type(NOT Allowed case 3)

In the given example, if the configuration database has an NOT_SUPPORTED_TELEMETRY_CLIENT table with the key "1234", it would correspond to the NOT_SUPPORTED_TELEMETRY_CLIENT_DS_LIST and NOT_SUPPORTED_TELEMETRY_CLIENT_SUB_LIST, this is an overlapping scenario

```yang ..... container NOT_SUPPORTED_TELEMETRY_CLIENT { list NOT_SUPPORTED_TELEMETRY_CLIENT_DS_LIST { // 1st list key "prefix";

    leaf prefix {
        type string {
            pattern ".*";
        }
    }

    leaf dst_addr {
        type ipv4-port;
    }
}

list NOT_SUPPORTED_TELEMETRY_CLIENT_SUB_LIST { // Negative case
    key "id";

    leaf id {
        type int32;
    }

    leaf dst_group {
        must "(contains(../../TELEMETRY_CLIENT_DS_LIST/prefix, current()))
        type int32;
    }
}

} ..... ```

**Example 5: keys with same number of elements and different type(NOT Allowed case 4)**

In the given example, if the configuration database has an NOT_SUPPORTED_TELEMETRY_CLIENT table with the key "1234|1234", it would correspond to the NOT_SUPPORTED_TELEMETRY_CLIENT_DS_LIST and NOT_SUPPORTED_TELEMETRY_CLIENT_SUB_LIST, this is an overlapping scenario

```yang ..... container NOT_SUPPORTED_TELEMETRY_CLIENT { list NOT_SUPPORTED_TELEMETRY_CLIENT_DS_LIST { // 1st list key "prefix name";

```
    leaf prefix {
        type string;
    }

    leaf name {
        type string;
    }

    leaf dst_addr {
        type ipv4-port;
    }
}

list NOT_SUPPORTED_TELEMETRY_CLIENT_SUB_LIST { // Negative case
    key "id name";

    leaf id {
        type int32;
    }

    leaf name {
        type int32;
    }

    leaf dst_group {
        must "(contains(../../TELEMETRY_CLIENT_DS_LIST/prefix, current()))
        type int32;
    }
}

} ..... ```
```

19. Add read-only nodes for state data using 'config false' statement. Define a separate top level container for state data. If state data is defined in other DB than CONFIG_DB, use extension 'sonic-ext:db-name' for defining the table present in other Redis DB. The default separator used in table key is "|", if it is different, use 'sonic-ext:key-delim {separator};' YANG extension. This step applies when SONiC YANG is used as Northbound YANG.

Example:

YANG

```yang container ACL_RULE { list ACL_RULE_LIST { .... .... container state { sonic-ext:db-name "APPL_DB"; //For example only sonic-ext:key-delim ":"; //For example only
```

```
        config false;
        description "State data";

        leaf MATCHED_PACKETS {
            type yang:counter64;
        }

        leaf MATCHED_OCTETS {
            type yang:counter64;
        }
    }
}
```

} ```

20. Define custom RPC for executing command like clear, reset etc. No configuration should change through such RPCs. Define 'input' and 'output' as needed, however they are optional. This step applies when SONiC YANG is used as Northbound YANG.

Example:

YANG

```yang container sonic-acl { .... .... rpc clear-stats {
input { leaf aclname { type string; }
```

```
        leaf rulename {
            type string;
        }
    }
}
```

} ```

21. Define Notification for sending out events generated in the system, e.g. link up/down or link failure event. This step applies when SONiC YANG is used as Northbound YANG.

Example:

YANG

```
yang module sonic-port { .... .... notification link_event { leaf
port { type leafref { path "../../PORT/PORT_LIST/ifname"; } } } }
```

APPENDIX

Sample SONiC ACL YANG

```yang module sonic-acl { namespace "http://github.com/sonic-net/sonic-acl"; prefix sacl; yang-version 1.1;

```
import ietf-yang-types {
    prefix yang;
}

import ietf-inet-types {
    prefix inet;
}

import sonic-common {
    prefix scommon;
}

import sonic-port {
    prefix prt;
}

import sonic-portchannel {
    prefix spc;
}

import sonic-mirror-session {
    prefix sms;
}

import sonic-pf-limits {
    prefix spf;
}

organization
    "SONiC";

contact
    "SONiC";

description
    "SONiC YANG ACL";

revision 2019-09-11 {
    description
        "Initial revision.";
}

container sonic-acl {
    container ACL_TABLE {
```
```

```
    list ACL_TABLE_LIST {
        key "table_name";

        leaf table_name {
            type string;
        }

        leaf policy_desc {
            type string {
                length 1..255 {
                    error-app-tag policy-desc-invalid-length;
                }
            }
        }

        leaf stage {
            type enumeration {
                enum INGRESS;
                enum EGRESS;
            }
        }

        leaf type {
            type enumeration {
                enum MIRROR;
                enum L2;
                enum L3;
                enum L3V6;
            }
        }

        leaf-list ports {
            type union {
                type leafref {
                    path "/prt:sonic-port/prt:PORT/prt:PORT_LIST/prt:i
                }
                type leafref {
                    path "/spc:sonic-portchannel/spc:PORTCHANNEL/spc:P
                }
            }
        }
    }
}

container ACL_RULE {
    list ACL_RULE_LIST {
        key "table_name rule_name";
        leaf table_name {
            type leafref {
                path "../../../ACL_TABLE/ACL_TABLE_LIST/table_name";
            }
            must "(/scommon:operation/scommon:operation != 'DELETE') o
```

```
            "(current()/../../../ACL_TABLE/ACL_TABLE_LIST[table_na
                error-message "Type not staisfied.";
            }
        }

        leaf rule_name {
            type string;
        }

        leaf PRIORITY {
            type uint16 {
                range "1..65535"{
                    error-message "Invalid ACL rule priority.";
            }
            }
        }

        leaf RULE_DESCRIPTION {
            type string;
        }

        leaf PACKET_ACTION {
            type enumeration {
                enum FORWARD;
                enum DROP;
                enum REDIRECT;
            }
        }

        leaf MIRROR_ACTION {
            type leafref {
                path "/sms:sonic-mirror-session/sms:MIRROR_SESSION/sms
            }
        }

        leaf IP_TYPE {
            type enumeration {
                enum ANY;
                enum IP;
                enum IPV4;
                enum IPV4ANY;
                enum NON_IPV4;
                enum IPV6ANY;
                enum NON_IPV6;
            }
        }

        leaf IP_PROTOCOL {
            type uint8 {
                range "1|2|6|17|46|47|51|103|115";
            }
        }
```

```
leaf ETHER_TYPE {
    type string {
        pattern "(0x88CC)|(0x8100)|(0x8915)|(0x0806)|(0x0800)|
            error-message "Invalid ACL Rule Ether Type";
            error-app-tag ether-type-invalid;
        }
    }
}

choice ip_src_dst {
    case ipv4_src_dst {
        when "boolean(IP_TYPE[.='ANY' or .='IP' or .='IPV4' or
        leaf SRC_IP {
            mandatory true;
            type inet:ipv4-prefix;
        }
        leaf DST_IP {
            mandatory true;
            type inet:ipv4-prefix;
        }
    }
    case ipv6_src_dst {
        when "boolean(IP_TYPE[.='ANY' or .='IP' or .='IPV6' or
        leaf SRC_IPV6 {
            mandatory true;
            type inet:ipv6-prefix;
        }
        leaf DST_IPV6 {
            mandatory true;
            type inet:ipv6-prefix;
        }
    }
}

choice src_port {
    case l4_src_port {
        leaf L4_SRC_PORT {
            type uint16;
        }
    }
    case l4_src_port_range {
        leaf L4_SRC_PORT_RANGE {
            type string {
                pattern "[0-9]{1,5}(-)[0-9]{1,5}";
            }
        }
    }
}

choice dst_port {
    case l4_dst_port {
```

```
                    leaf L4_DST_PORT {
                        type uint16;
                    }
                }
                case l4_dst_port_range {
                    leaf L4_DST_PORT_RANGE {
                        type string {
                            pattern "[0-9]{1,5}(-)[0-9]{1,5}";
                        }
                    }
                }
            }

            leaf TCP_FLAGS {
                type string {
                    pattern "0[xX][0-9a-fA-F]{2}[/]0[xX][0-9a-fA-F]{2}";
                }
            }

            leaf DSCP {
                type uint8;
            }
        }
    }
}

}
```