

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **Policer APIs** |
| **Authors** | **DELL** |
| **Status** | **In review** |
| **Type** | **Standards Track** |
| **Created** | **02/19/2015** |
| **SAI-Version** | **V0.9.3** |

**Contents**

[List of Changes i](#_Toc418687209)

[1 Overview 1](#_Toc418687210)

[2 Specification 1](#_Toc418687211)

[2.1 Changes to sai.h 1](#_Toc418687212)

[2.2.1New File saipolicer.h 1](#_Toc418687213)

[1) Create Policer 3](#_Toc418687214)

[2) Delete Policer 4](#_Toc418687215)

[3) Set Attribute to Policer 4](#_Toc418687216)

[4) Get Attribute to Policer 4](#_Toc418687217)

[5) Get Policer Stats 5](#_Toc418687218)

[6) Method Table for Policer 5](#_Toc418687219)

[2.3 Changes to saiport.h 5](#_Toc418687220)

[2.3.1 New Attributes 5](#_Toc418687221)

[2.4 Modified Attributes 5](#_Toc418687222)

[2.5 Changes to saiacl.h 5](#_Toc418687223)

[2.6 Example configurations 6](#_Toc418687224)

[2.6.1 Example to create policer 6](#_Toc418687225)

[2.6.2 Ingress Policer: Apply policer to ingress ACL 6](#_Toc418687226)

[2.6.3 Egress rate limit: Apply policer to Egress ACL 7](#_Toc418687227)

[2.6.4 Apply policer to PORT 7](#_Toc418687228)

[2.6.5 Remove policer from PORT 7](#_Toc418687229)

[2.6.6 Policer counters 7](#_Toc418687230)

[2.6.7 **Policer for Storm control functionality** 8](#_Toc418687231)

[2.6.8 Enable storm controls on a port 9](#_Toc418687232)

[3 Appendix 10](#_Toc418687233)

[3.1 To be considered for next version of policer 10](#_Toc418687234)

[3.1.1 Color Source: 10](#_Toc418687235)

[3.1.2 Actions: 10](#_Toc418687236)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.2 | Proposal for Policer |  | 2/19/2015 |

License

© 2014 Microsoft Corporation, Dell Inc., Facebook, Inc, Broadcom Corporation, Intel Corporation, Mellanox Technologies Ltd.

As of September 9, 2014, the following persons or entities have made this Specification available under the Open Web Foundation Final Specification Agreement (OWFa 1.0), which is available at <http://www.openwebfoundation.org/legal/the-owf-1-0-agreements/owfa-1-0>

Microsoft Corporation, Dell Inc., Facebook, Inc, Intel Corporation, Mellanox Technologies Ltd.

You can review the signed copies of the Open Web Foundation Agreement Version 1.0 for this Specification at <http://opencompute.org/licensing/>, which may also include additional parties to those listed above.

Your use of this Specification may be subject to other third party rights. THIS SPECIFICATION IS PROVIDED "AS IS." The contributors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, noninfringement, fitness for a particular purpose, or title, related to the Specification. The entire risk as to implementing or otherwise using the Specification is assumed by the Specification implementer and user. IN NO EVENT WILL ANY PARTY BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION OR ITS GOVERNING AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE FOLLOWING IS A LIST OF MERELY REFERENCED TECHNOLOGY: Microprocessor technology, semiconductor manufacturing technology, operating system technology (including without limitation networking operating system technology), emulation technology, graphics technology, video technology, integrated circuit packaging technology and the like, compiler technologies, object oriented technology, optical/RF communications technology including chip I/O and driver technology, bus technology, memory chip technology (including, without limitation, NAND memory, NOR memory, resistive RAM (RRAM), seek scan probe (SSP) memory, nonvolatile memory (including without limitation, memory based on chalcogenide materials, phase change memory (PCM), one or more stacked layers of memory cells, embedded PCM memories, non-volatile cache memory, solid state drives, SRAM, embedded DRAM, ferro-electric memory, and polymer memory)) and/or health-related and medical technology. IMPLEMENTATION OF THESE TECHNOLOGIES MAY BE SUBJECT TO THEIR OWN LEGAL TERMS.

# Overview

Traffic policers enable you to control the maximum rate of traffic sent or received on a device interface. The policer is a separate object which can be created/removed and set and get operations can be performed on its attributes.

The policer will be associated to ACL’s, port etc. Storm control is also policer for the defined set of packet types. Policer can be associated to port to set the storm control. Some NPU’s supports normal policer and storm control policer on a same port. Behavior depends on the NPU.

**Terms used:**

CIR - Committed Information Rate.

CBS – Committed Burst Size.

PBS/EBS – Peak Burst Size.

PIR/EIR – Peak Information Rate.

**Policer counter:**  Policer counter is separate h/w object in some NPU’s. Policer counter will be created and associated policer. By using policer counter statistics can be retrieved further.

In case of policer counter h/w object is not supported and policer associated to ACL then statistics can get from the regular acl counter actions by applications.

# Specification

## 2.1 Changes to sai.h

typedef enum \_sai\_api\_t {

SAI\_API\_POLICER= 13, /\* sai\_api\_t \*

}

### 2.2.1New File saipolicer.h

/\* Meter type \*/

typedef enum \_sai\_meter\_type\_t

{

SAI\_METER\_TYPE\_PACKETS,

SAI\_METER\_TYPE\_BYTES,

} sai\_meter\_type\_t;

typedef enum \_sai\_policer\_mode\_t,

{

/\* RFC 2697, Single Rate Three color marker, CIR, CBS and PBS, G, Y and R\*/

SAI\_POLICER\_MODE\_Sr\_TCM,

/\* RFC 2698, Two Rate Three color marker, CIR, CBS , PIR and PBS, G, Y and R\*/

SAI\_POLICER\_MODE\_Tr\_TCM,

/\* Storm control mode

Single Rate Two color CIR, CBS, G and R \*/

SAI\_POLICER\_MODE\_STORM\_CONTROL,

} sai\_policer\_mode\_t;

/\* Policer color source \*/

typedef enum \_sai\_policer\_color\_source\_t

{

SAI\_POLICER\_COLOR\_SOURCE\_BLIND,

SAI\_POLICER\_COLOR\_SOURCE\_AWARE,

} sai\_policer\_color\_source\_t;

/\* Policer Attributes \*/

typedef enum \_sai\_policer\_attr\_t

{

/\* READ-ONLY \*/

/\* READ-WRITE \*/

/\* Policer Meter Type [sai\_meter\_type\_t ]

\*MANDATORY\_ON\_CREATE, CREATE\_ONLY \*/

SAI\_POLICER\_ATTR\_METER\_TYPE,

/\* Policer mode [sai\_policer\_mode\_t],

Default SAI\_POLICER\_MODE\_Tr\_TCM \*/

SAI\_POLICER\_ATTR\_MODE,

/\* Policer Color Source [sai\_policer\_color\_source\_t],

Default is SAI\_POLICER\_COLOR\_SOURCE\_AWARE \*/

SAI\_POLICER\_ATTR\_COLOR\_SOURCE,

/\* Commited burst size bytes/packets based on SAI\_POLICER\_ATTR\_METER\_TYPE.

[sai\_uint64\_t] \*/

SAI\_POLICER\_ATTR\_CBS,

/\* Commited information rate BPS/PPS based on SAI\_POLICER\_ATTR\_METER\_TYPE.

[sai\_uint64\_t] \*/

SAI\_POLICER\_ATTR\_CIR,

/\* Peak burst size bytes/packets based on SAI\_POLICER\_ATTR\_METER\_TYPE.

[sai\_uint64\_t] \*/

SAI\_POLICER\_ATTR\_PBS,

/\* Peak information rate BPS/PPS based on SAI\_POLICER\_ATTR\_METER\_TYPE \*/

/\* Mandatory only for SAI\_POLICER\_MODE\_Tr\_TCM. [sai\_uint64\_t] \*/

SAI\_POLICER\_ATTR\_PIR,

/\* Action to take for Green color packets [sai\_packet\_action\_t],

Default action SAI\_PACKET\_ACTION\_FORWARD \*/

SAI\_POLICER\_ATTR\_GREEN\_PACKET\_ACTION,

/\* Action to take for Yellow color packets [sai\_packet\_action\_t],

Default action SAI\_PACKET\_ACTION\_FORWARD \*/

SAI\_POLICER\_ATTR\_YELLOW\_PACKET\_ACTION,

/\* Action to take for RED color packets [sai\_packet\_action\_t],

Default action SAI\_PACKET\_ACTION\_FORWARD.

Storm control action should be used as SAI\_POLICER\_ATTR\_RED\_PACKET\_ACTION \*/

SAI\_POLICER\_ATTR\_RED\_PACKET\_ACTION,

/\* Enable/disable counter [sai\_s32\_list\_t of sai\_policer\_stat\_counter\_t].

\* Default[disabled], Modify List Needs full new set\*/

    SAI\_POLICER\_ATTR\_ENABLE\_COUNTER\_LIST,

/\* -- \*/

/\* Custom range base value \*/

SAI\_POLICER\_ATTR\_CUSTOM\_RANGE\_BASE = 0x10000000

} sai\_policer\_attr\_t;

typedef enum \_sai\_policer\_stat\_counter\_t

{

/\*\* get/set packet count [uint64\_t] \*/

SAI\_POLICER\_STAT\_PACKETS,

/\*\* get/set byte count [uint64\_t] \*/

SAI\_POLICER\_STAT\_ATTR\_BYTES,

/\*\* get/set green packet count [uint64\_t] \*/

SAI\_POLICER\_STAT\_GREEN\_PACKETS,

/\*\* get/set green byte count [uint64\_t] \*/

SAI\_POLICER\_STAT\_GREEN\_BYTES,

/\*\* get/set yellow packet count [uint64\_t] \*/

SAI\_POLICER\_STAT\_YELLOW\_PACKETS,

/\*\* get/set yellow byte count [uint64\_t] \*/

SAI\_POLICER\_STAT\_YELLOW\_BYTES,

/\*\* get/set red packet count [uint64\_t] \*/

SAI\_POLICER\_STAT\_RED\_PACKETS,

/\*\* get/set red byte count [uint64\_t] \*/

SAI\_POLICER\_STAT\_RED\_BYTES,

/\* -- \*/

/\* Custom range base value \*/

SAI\_POLICER\_STAT\_CUSTOM\_RANGE\_BASE = 0x10000000

}sai\_policer\_stat\_counter\_t;

## Create Policer

/\*

\* Routine Description:

\* Create Policer

\*

\* Arguments:

\* [out] policer\_id - the policer id

\* [in] attr\_count - number of attributes

\* [in] attr\_list - array of attributes

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_create\_policer\_fn)(

\_Out\_ sai\_object\_id\_t \*policer\_id,

\_In\_ int attr\_count,

\_In\_ const sai\_attribute\_t \*attr\_list

);

## Delete Policer

/\*

\* Routine Description:

\* Delete policer

\*

\* Arguments:

\* [in] policer\_id - Policer id

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_policer\_fn)(

\_In\_ sai\_object\_id\_t policer\_id

);

## Set Attribute to Policer

/\*

\* Routine Description:

\* Set Policer attribute

\*

\* Arguments:

\* [in] policer\_id - Policer id

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_policer\_attribute\_fn)(

\_In\_ sai\_object\_id\_t policer\_id,

\_In\_ const sai\_attribute\_t \*attr

);

## Get Attribute to Policer

/\*

\* Routine Description:

\* Get Policer attribute

\*

\* Arguments:

\* [in] policer\_id - policer id

\* [in] attr\_count - number of attributes

\* [Out] attr\_list - array of attributes

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_policer\_attribute\_fn)(

\_In\_ sai\_object\_id\_t policer\_id,

\_In\_ int attr\_count,

\_Out\_ sai\_attribute\_t \*attr\_list

);

## Get Policer Stats

typedef sai\_status\_t (\*sai\_get\_policer\_stats\_fn)(

    \_In\_ sai\_object\_id\_t policer\_id,

    \_In\_ const sai\_policer\_stat\_counter\_t \*counter\_ids,

    \_In\_ uint32\_t number\_of\_counters,

    \_Out\_ uint64\_t\* counters

    );

## Method Table for Policer

/\*

\* Policer methods table retrieved with sai\_api\_query()

\*/

typedef struct \_sai\_policer\_api\_t

{

sai\_create\_policer\_fn create\_policer;

sai\_remove\_policer\_fn remove\_policer;

sai\_set\_policer\_attribute\_fn set\_policer\_attribute;

sai\_get\_policer\_attribute\_fn get\_policer\_attribute;

sai\_get\_policer\_stats\_fn get\_policer\_stats;

} sai\_policer\_api\_t;

## Changes to saiport.h

### New Attributes

typedef enum \_sai\_port\_attr\_t

{

..

..

/\* Attach/Detach policer to port [sai\_object\_id\_t],

Policer id = SAI\_NULL\_OBJECT\_ID to disable policer on port \*/

SAI\_PORT\_ATTR\_POLICER\_ID,

} sai\_port\_attr\_t;

## Modified Attributes

The following attributes will be modified from the port attributes list. A policer-id will be associated to achieve storm control functionality.

/\* Enable flood (unknown unicast or unknown multicast)

storm control policer on port [sai\_object\_id\_t]

Policer id = SAI\_NULL\_OBJECT\_ID to disable policer on port \*/

SAI\_PORT\_ATTR\_FLOOD\_STORM\_CONTROL\_POLICER\_ID,

/\* Enable broadcast storm control policer on port [sai\_object\_id\_t]

Policer id = SAI\_NULL\_OBJECT\_ID to disable policer on port \*/

SAI\_PORT\_ATTR\_BROADCAST\_STORM\_CONTROL\_POLICER\_ID,

/\* Enable multicast storm control policer on port [sai\_object\_id\_t]

Policer id = SAI\_NULL\_OBJECT\_ID to disable policer on port \*/

SAI\_PORT\_ATTR\_MULTICAST\_STORM\_CONTROL\_POLICER\_ID,

## Changes to saiacl.h

typedef enum \_sai\_acl\_counter\_attr\_t

{

SAI\_ACL\_COUNTER\_ATTR\_TABLE\_ID,

/\* Enable counter [sai\_s32\_list\_t of sai\_acl\_counter\_attr\_t].

\* (MANDATORY\_ON\_CREATE | CREATE\_SET), with atleast one counter type enable

\* Modify List Needs full new set \*/

    SAI\_ACL\_COUNTER\_ATTR\_**ENABLE**\_COUNT\_LIST,

/\* Below Counter Types for Enable/get/set \*/

/\* Packet count [uint64\_t] \*/

SAI\_ACL\_COUNTER\_ATTR\_PACKETS,

/\* Byte count [uint64\_t] \*/

SAI\_ACL\_COUNTER\_ATTR\_BYTES,

/\* Green packet count [uint64\_t] \*/

SAI\_ACL\_COUNTER\_ATTR\_GREEN\_PACKETS,

/\* Green byte count [uint64\_t] \*/

SAI\_ACL\_COUNTER\_ATTR\_GREEN\_BYTES,

/\* Yellow packet count [uint64\_t] \*/

SAI\_ACL\_COUNTER\_ATTR\_YELLOW\_PACKETS,

/\* Yellow byte count [uint64\_t] \*/

SAI\_ACL\_COUNTER\_ATTR\_YELLOW\_BYTES,

/\* Red packet count [uint64\_t] \*/

SAI\_ACL\_COUNTER\_ATTR\_RED\_PACKETS,

/\* Red byte count [uint64\_t] \*/

SAI\_ACL\_COUNTER\_ATTR\_RED\_BYTES,

} sai\_acl\_counter\_attr\_t;

## Example configurations

### Example to create policer

Create a policer with mode single rate tricolor, type packets, color source as color aware. Below steps describe the creation of the policer.

#### Step-1: create policer

sai\_object\_id\_t policer\_id;

sai\_attribute\_t attr\_list[6];

int attr\_count = 6;

attr\_list[0].id = SAI\_POLICER\_ATTR\_METER\_TYPE;

attr\_list[0].value.s32 = SAI\_METER\_TYPE\_PACKETS;

attr\_list[1].id = SAI\_POLICER\_ATTR\_MODE;

attr\_list[1].value.s32 = SAI\_POLICER\_MODE\_Sr\_TCM;

attr\_list[2].id = SAI\_POLICER\_ATTR\_COLOR\_SOURCE;

attr\_list[2].value.s32 = SAI\_POLICER\_COLOR\_SOURCE\_BLIND;

attr\_list[3].id = SAI\_POLICER\_ATTR\_CBS;

attr\_list[3].value.s32 = 50;

attr\_list[4].id = SAI\_POLICER\_ATTR\_CIR;

attr\_list[4].value.s32 = 100;

attr\_list[5].id = SAI\_POLICER\_ATTR\_PBS;

attr\_list[5].value.s32 = 75;

sai\_create\_policer\_fn (&policer\_id, attr\_count, &attr\_list);

### Ingress Policer: Apply policer to ingress ACL

This created policer will be applied as an action to ACL rule in case of a normal policer.

sai\_object\_id\_t acl\_entry\_id;

sai\_attribute\_t sai\_acl\_attr\_set;

sai\_acl\_attr\_set.id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_SET\_POLICER ;

sai\_acl\_attr\_set.value.oid = policer\_id;

sai\_set\_acl\_entry\_attribute\_fn (acl\_entry\_id, &sai\_acl\_attr\_set);

### Egress rate limit: Apply policer to Egress ACL

This created policer will be applied as an action to ACL rule in case of a normal policer.

sai\_attribute\_t sai\_acl\_attr\_set;

sai\_acl\_attr\_set.id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_SET\_POLICER ;

sai\_acl\_attr\_set.value.oid = policer\_id;

sai\_set\_acl\_entry\_attribute\_fn (acl\_entry\_id, &sai\_acl\_attr\_set);

### Apply policer to PORT

This created policer will be applied as an action to ACL rule in case of a normal policer.

sai\_object\_id\_t port\_id;

sai\_attribute\_t sai\_port\_attr\_set;

sai\_port\_attr\_set.id = SAI\_PORT\_ATTR\_POLICER\_ID ;

sai\_port\_attr\_set.value.oid = policer\_id;

sai\_set\_port\_attribute\_fn (port\_id, &sai\_acl\_attr\_set);

### Remove policer from PORT

This created policer will be applied as an action to ACL rule in case of a normal policer.

sai\_attribute\_t sai\_port\_attr\_set;

sai\_object\_id\_t port\_id;

sai\_port\_attr\_set.id = SAI\_PORT\_ATTR\_POLICER\_ID ;

sai\_port\_attr\_set.value.oid = SAI\_NILL\_OBJECT\_ID;

sai\_set\_port\_attribute\_fn (port, &sai\_acl\_attr\_set);

### Policer counters

#### Example for NPU supports H/w Policer counter

##### Step-1: Enable policer counter

sai\_attribute\_t sai\_acl\_attr\_set;

sai\_acl\_attr\_set.id = SAI\_POLICER\_ATTR\_ENABLE\_COUNTER\_LIST ;

sai\_acl\_attr\_set.value.s32list.count = 3;

sai\_acl\_attr\_set.value.s32list.list = calloc(3, sizeof(int32\_t));

sai\_acl\_attr\_set.value.s32list.list[0] = SAI\_POLICER\_STAT\_GREEN\_PACKETS;

sai\_acl\_attr\_set.value.s32list.list[1] = SAI\_POLICER\_STAT\_YELLOW\_PACKETS;

sai\_acl\_attr\_set.value.s32list.list[2] = SAI\_POLICER\_STAT\_RED\_PACKETS;

sai\_set\_policer\_attribute(policer\_id, &sai\_acl\_attr\_set);

##### Step-2: Retrieve Policer counter values

sai\_policer\_stat\_counter\_t policer\_stat\_id[3] ;

uint64\_t policer\_counters[3] = {0,0,0} ;

policer\_stat\_id[0] = SAI\_POLICER\_STAT\_GREEN\_PACKETS ;

policer\_stat\_id[1] = SAI\_POLICER\_STAT\_YELLOW\_PACKETS ;

policer\_stat\_id[2] = SAI\_POLICER\_STAT\_RED\_PACKETS ;

sai\_get\_policer\_stats(policer\_id, policer\_stat\_id, 3, policer\_counters);

#### Example for NPU does not supports H/w Policer counter

##### Step-1: Check for policer counter support.

sai\_attribute\_t sai\_acl\_attr\_set;

sai\_acl\_attr\_set.id = SAI\_POLICER\_ATTR\_ENABLE\_COUNTER\_LIST ;

sai\_acl\_attr\_set.value.s32list.count = 3;

sai\_acl\_attr\_set.value.s32list.list = calloc(3, sizeof(int32\_t));

sai\_acl\_attr\_set.value.s32list.list[0] = SAI\_POLICER\_STAT\_GREEN\_PACKETS;

sai\_acl\_attr\_set.value.s32list.list[1] = SAI\_POLICER\_STAT\_YELLOW\_PACKETS;

sai\_acl\_attr\_set.value.s32list.list[2] = SAI\_POLICER\_STAT\_RED\_PACKETS;

sai\_set\_policer\_attribute(policer\_id, &sai\_acl\_attr\_set);

/\* SAI returns will return SAI\_STATIS\_NPT\_SUPPORTED \*/

##### Step-2: Apply policer to ACL.

sai\_object\_id\_t acl\_entry\_id;

sai\_attribute\_t sai\_acl\_attr\_set;

sai\_acl\_attr\_set.id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_SET\_POLICER ;

sai\_acl\_attr\_set.value.oid = policer\_id;

sai\_set\_acl\_entry\_attribute\_fn (acl\_entry\_id, &sai\_acl\_attr\_set);

##### Step-3: Create ACL Counter

sai\_object\_id\_t acl\_counter\_id;

sai\_object\_id\_t acl\_table\_id;

sai\_attribute\_t attr\_list;

sai\_s32\_list\_t count\_type\_list;

count\_type\_list.count = 2;

count\_type\_list.list = calloc(count\_type\_list.count, sizeof(int));

count\_type\_list.list[0] = SAI\_ACL\_COUNTER\_ATTR\_GREEN\_PACKETS;

count\_type\_list.list[1] = SAI\_ACL\_COUNTER\_ATTR\_YELLOW\_PACKETS;

int attr\_count = 2;

attr\_list[0].id = SAI\_ACL\_COUNTER\_ATTR\_TABLE\_ID;

attr\_list[0].value.oid = acl\_table\_id;

attr\_list[1].id = SAI\_ACL\_COUNTER\_ATTR\_ENABLE\_COUNT\_LIST;

attr\_list[1].value. s32list = count\_type\_list;

sai\_create\_acl\_counter\_fn (&acl\_counter\_id, attr\_count, &attr\_list);

##### Step-4: Apply ACL counter to ACL entry

sai\_object\_id\_t acl\_entry\_id;

sai\_attribute\_t sai\_acl\_attr\_set;

sai\_acl\_attr\_set.id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_SET\_COUNTER ;

sai\_acl\_attr\_set.value.oid = acl\_counter\_id;

sai\_set\_acl\_entry\_attribute\_fn (acl\_entry\_id, &sai\_acl\_attr\_set);

##### Step-5: Retrieve ACL counter values

sai\_attribute\_t attr\_list ;

int attr\_count = 2;

attr\_list[0].id = SAI\_ACL\_COUNTER\_ATTR\_GREEN\_PACKETS;

attr\_list[0].value.u64 = 0;

attr\_list[1].id = SAI\_ACL\_COUNTER\_ATTR\_YELLOW\_PACKETS;

attr\_list[1].value.u64 = 0;

sai\_get\_acl\_counter\_attribute\_fn (acl\_counter\_id, 3, &sai\_acl\_attr\_set);

### **Policer for Storm control functionality**

sai\_object\_id\_t flood\_policer\_id;

sai\_object\_id\_t multicast\_policer\_id;

sai\_object\_id\_t broadcast\_policer\_id;

sai\_attribute\_t attr\_list;

int attr\_count = 6;

attr\_list[0].id = SAI\_POLICER\_ATTR\_METER\_TYPE;

attr\_list[0].value.s32 = SAI\_METER\_TYPE\_PACKETS;

attr\_list[1].id = SAI\_POLICER\_ATTR\_MODE;

attr\_list[1].value.s32 = SAI\_POLICER\_MODE\_STORM\_CONTROL;

attr\_list[2].id = SAI\_POLICER\_ATTR\_COLOR\_SOURCE;

attr\_list[2].value.s32 = SAI\_POLICER\_COLOR\_SOURCE\_BLIND;

attr\_list[3].id = SAI\_POLICER\_ATTR\_CBS;

attr\_list[3].value.s32 = 50;

attr\_list[4].id = SAI\_POLICER\_ATTR\_CIR;

attr\_list[4].value.s32 = 100;

sai\_create\_policer\_counter\_fn (&flood\_policer\_id, attr\_count, &attr\_list);

attr\_list[0].id = SAI\_POLICER\_ATTR\_METER\_TYPE;

attr\_list[0].value.s32 = SAI\_METER\_TYPE\_PACKETS;

attr\_list[1].id = SAI\_POLICER\_ATTR\_MODE;

attr\_list[1].value.s32 = SAI\_POLICER\_MODE\_STORM\_CONTROL;

attr\_list[2].id = SAI\_POLICER\_ATTR\_COLOR\_SOURCE;

attr\_list[2].value.s32 = SAI\_POLICER\_COLOR\_SOURCE\_BLIND;

attr\_list[3].id = SAI\_POLICER\_ATTR\_CBS;

attr\_list[3].value.s32 = 50;

attr\_list[4].id = SAI\_POLICER\_ATTR\_CIR;

attr\_list[4].value.s32 = 100;

sai\_create\_policer\_counter\_fn (&multicast\_policer\_id, attr\_count, &attr\_list);

attr\_list[0].id = SAI\_POLICER\_ATTR\_METER\_TYPE;

attr\_list[1].value.s32 = SAI\_METER\_TYPE\_PACKETS;

attr\_list[1].id = SAI\_POLICER\_ATTR\_MODE;

attr\_list[1].value.s32 = SAI\_POLICER\_MODE\_STORM\_CONTROL;

attr\_list[2].id = SAI\_POLICER\_ATTR\_COLOR\_SOURCE;

attr\_list[2].value.s32 = SAI\_POLICER\_COLOR\_SOURCE\_BLIND;

attr\_list[3].id = SAI\_POLICER\_ATTR\_CBS;

attr\_list[3].value.s32 = 50;

attr\_list[4].id = SAI\_POLICER\_ATTR\_CIR;

attr\_list[4].value.s32 = 100;

sai\_create\_policer\_counter\_fn (&broadcast\_policer\_id, attr\_count, &attr\_list);

### Enable storm controls on a port

sai\_object\_id\_t port\_id = 0;

sai\_attribute\_t sai\_port\_attr\_set;

sai\_port\_attr\_set.id = SAI\_PORT\_ATTR\_FLOOD\_STORM\_CONTROL;

sai\_port\_attr\_set.value.oid = flood\_policer\_id;

sai\_set\_port\_attribute(port\_id, &sai\_attr\_set);

sai\_port\_attr\_set.id = SAI\_PORT\_ATTR\_MULTICAST\_STORM\_CONTROL;

sai\_port\_attr\_set.value.oid = multicast\_policer\_id;

sai\_set\_port\_attribute(port\_id, &sai\_attr\_set);

sai\_port\_attr\_set.id = SAI\_PORT\_ATTR\_BROADCAST\_STORM\_CONTROL;

sai\_port\_attr\_set.value.oid = multicast\_policer\_id;

sai\_set\_port\_attribute\_fn(port\_id, &sai\_attr\_set);

# Appendix

## To be considered for next version of policer

### Color Source:

Initial version of proposal is going with minimum requirements for policer. And color source aware is assumed by any previous tables. Functionality of selecting the color source are extended by adding the

typedef enum \_sai\_policer\_color\_source\_t

{

SAI\_POLICER\_COLOR\_SOURCE\_BLIND,

SAI\_POLICER\_COLOR\_SOURCE\_AWARE,

SAI\_POLICER\_COLOR\_SOURCE\_DSCP,

SAI\_POLICER\_COLOR\_SOURCE\_DOT1P

} sai\_policer\_color\_source\_t;

### Actions:

SAI\_POLICER\_ATTR\_YELLOW\_PACKET\_ACTION\_NEW\_DSCP,

SAI\_POLICER\_ ATTR\_YELLOW\_PACKET \_ACTION\_NEW\_ECN,

SAI\_POLICER\_ATTR\_YELLOW\_PACKET\_ACTION\_NEW\_VLAN\_PRIORITY,

SAI\_POLICER\_ATTR\_YELLOW\_PACKET\_ACTION\_NEW\_CNG,

SAI\_POLICER\_ATTR\_RED\_PACKET\_ACTION\_NEW\_DSCP,

SAI\_POLICER\_ATTR\_RED\_PACKET\_ACTION\_NEW\_ECN,

SAI\_POLICER\_ATTR\_RED\_PACKET\_ACTION\_NEW\_VLAN\_PRIORITY,

SAI\_POLICER\_ATTR\_RED\_PACKET\_ACTION\_NEW\_CNG,

/\* sai\_policer\_dscp\_mkdn\_map

    The map is indexed by the current value of the frame's   DSCP field with the entry being the new DSCP value to mark  down  \*/

SAI\_POLICER\_ATTR\_YELLOW\_PACKET\_MARK\_DOWN\_DSCP,

SAI\_POLICER\_ATTR\_RED\_PACKET\_MARK\_DOWN\_DSCP,