

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **QOS Queue API’s** |
| **Authors** | **DELL** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **03/04/2015** |
| **SAI-Version** | **V0.9.3** |

**Contents**

[List of Changes i](#_Toc419197953)

[1 Overview 1](#_Toc419197954)

[1.1 Queuing models 1](#_Toc419197955)

[1.1.1 Diffserv Queuing 1](#_Toc419197956)

[1.2 Application Requirements 1](#_Toc419197957)

[1.2.1 Configurable Number of Queues per Port 2](#_Toc419197958)

[1.2.2 H/w Egress Queues 2](#_Toc419197959)

[1.3 SAI Queue Representation 2](#_Toc419197960)

[1.3.1 Queue Object 2](#_Toc419197961)

[2 Specification 2](#_Toc419197962)

[2.1.1 Changes to saiswitch.h 2](#_Toc419197963)

[2.1.2 Changes to saiport.h 2](#_Toc419197964)

[2.1.3 Changes to saitypes.h 3](#_Toc419197965)

[2.1.4 Changes to saiqos.h 3](#_Toc419197966)

[1) Set attribute to queue 3](#_Toc419197967)

[2) Get attribute of queue 4](#_Toc419197968)

[2.1.5 Queue statistics 4](#_Toc419197969)

[2.2 Configuration examples 6](#_Toc419197970)

[2.2.1 Step-1: Get number of queue supported on port 6](#_Toc419197971)

[2.2.2 Step-2: Get list of SAI queue object id’s on port 6](#_Toc419197972)

[**2.2.3** Step 3: Map Application queues to sai queues 6](#_Toc419197973)

[2.2.4 Applying the wred to application unicast queue 0 8](#_Toc419197974)

[2.2.5 Applying the schedulers to Application multicast queue 3 8](#_Toc419197975)

[2.2.6 Map traffic class -> queues (8 queue model) 8](#_Toc419197976)

[2.2.7 Example Configurable Number of Queues per Port 8](#_Toc419197977)

[3 Appendix 9](#_Toc419197978)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.2 | Proposal for Queue |  | 03/04/15 |

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

QoS enables you to provide better service to certain flows. Queuing determines how packets are queued in various queues in the egress interface.

## Queuing models

### Diffserv Queuing

Diffserv is traditional model where differentiated services are provided by mapping the certain traffic types to egress queues. This will accomplish generally by defining the traffic classes (TCs) for packets and assign traffic to egress queue on per-TC basis and optionally other packet fields sources like VLAN, dot1p etc.

#### Traffic Class:

Traffic class is represents the normalized view of qos within the system. Traffic classes are derived for data packet in inbound port by QOS attributes:

* Dot1p value
* DSCP value
* Ingress default port setting
* ACL

Traffic Class (TC): For Internal priority (BCM) or Switch priority (Intel) or priority in (MX).

#### Transmit Queues

A queue is a location (or buffer) containing a finite number of items waiting for an action or service. In networking, a queue is the place where packets (our units) wait to be transmitted by the hardware (the service).

Transmission queues are configurable for individual ports to shape traffic based on its traffic class. Data packets are assigned to queues based on their traffic class, then sent as scheduled by port and transmit settings. Traffic class is often referred to as a queue, it is not necessarily a single queue. For example, an implementation might organize the traffic class as a set of queues link unicast or multicast with each queue containing the frames from a priority, VLAN or a source port.

##### Unicast Queues:

Queues used to schedule/buffer unicast packets.

##### Multicast Queues:

Queues used to schedule/buffer multicast packets.

## Application Requirements

### Configurable Number of Queues per Port

ASIC’s may support fixed/configurable number of queues per port. Application needs to control of number of queues per port based on requirement. Traffic needs to be allowed only to queues created by applications. This will give control to application to save global QOS resources like buffers etc.

### H/w Egress Queues

Traditionally, switches support 8 egress queues per output port, each servicing one IEEE 802.1p Dot1p. Some ASIC’s support separate unicast and multicast egress queues. This support allows control over contending for system resources within the same CoS and provides more fairness between unicast and multicast. Through configuration, the user can control the amount of egress port bandwidth for each of the 16 egress queues.

Applications requires the schedule/transmit settings for unicast/multicast separately for the platforms supports separate physical queues.

## SAI Queue Representation

This proposal for SAI QOS covers only the queue representation and essential attributes for queue.

### Queue Object

SAI defines the queue object as container to apply the attributes which are control queue.

Queue Attributes:

* Buffer limit profile
* WRED profile
* Scheduler profile

# Specification

### Changes to saiswitch.h

#### New attributes

typedef enum \_sai\_switch\_attr\_t

{

..

..

/\* READ-ONLY \*/

/\*\* Maximum traffic classes limit\*/

SAI\_SWITCH\_ATTR\_QOS\_MAX\_NUMBER\_OF\_TRAFFIC\_CLASSES,

} sai\_switch\_attr\_t;

### Changes to saiport.h

#### New attributes

typedef enum \_sai\_port\_attr\_t

{

..

..

/\* READ-ONLY \*/

/\* Get the port list [sai\_object\_list\_t] \*/

SAI\_PORT\_ATTR\_QOS\_QUEUE\_LIST,

} sai\_port\_attr\_t;

### Changes to saitypes.h

### Changes to saiqos.h

#### New definitions

typedef enum \_sai\_qos\_queue\_type\_t {

/\* H/w Queue for all types of traffic \*/

SAI\_QOS\_QUEUE\_TYPE\_NONE,

/\* H/w Unicast Queue \*/

SAI\_QOS\_QUEUE\_TYPE\_UNICAST,

/\* H/w Multicast (Broadcast, Unknown unicast, Multicast) Queue \*/

SAI\_QOS\_QUEUE\_TYPE\_MULTICAST,

**} s**ai\_qos\_queue\_type\_t;

typedef enum \_sai\_qos\_queue\_attr\_t

{

/\* READ-ONLY \*/

/\* Queue type [**s**ai\_qos\_queue\_type\_t] \*/

SAI\_QOS\_QUEUE\_ATTR\_TYPE,

/\* READ-WRITE \*/

/\* Drop Type [sai\_qos\_drop\_type\_t],

Default (TAIL DROP) \*/

SAI\_QOS\_QUEUE\_ATTR\_DROP\_TYPE,

/\* Attach WRED ID to queue [sai\_wred\_id\_t]

Mandatory when SAI\_QOS\_QUEUE\_ATTR\_DROP\_TYPE = SAI\_QOS\_DROP\_TYPE\_WRED \*/

SAI\_QOS\_QUEUE\_ATTR\_WRED\_PROFILE\_ID,

SAI\_QOS\_QUEUE\_ATTR\_BUFFER\_PROFILE\_ID,

SAI\_QOS\_QUEUE\_ATTR\_SCHEDULER\_PROFILE\_ID,

/\* -- \*/

/\* Custom range base value \*/

SAI\_QOS\_QUEUE\_ATTR\_CUSTOM\_RANGE\_BASE = 0x10000000

} sai\_qos\_queue\_attr\_t;

## Set attribute to queue

/\*

\* Routine Description:

\* Set queue attribute

\*

\* Arguments:

\* [in] queue\_id – the queue id

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_qos\_queue\_attribute\_fn)(

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

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

);

## Get attribute of queue

/\*

\* Routine Description:

\* Get queue attribute

\*

\* Arguments:

\* [in] queue\_id – queue counter 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\_qos\_queue\_attribute\_fn)(

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

\_In\_ uint32\_t attr\_count,

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

);

### Queue statistics

typedef enum \_sai\_qos\_queue\_stat\_counter\_t

{

/\* get/set tx packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_PACKETS,

/\* get/set tx bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_BYTES,

/\* get/set dropped packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_DROPPED\_PACKETS,

/\* get/set dropped bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_DROPPED\_BYTES,

/\* get/set green color tx packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_GREEN\_PACKETS,

/\* get/set green color tx bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_GREEN\_BYTES,

/\* get/set green color dropped packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_GREEN\_DROPPED\_PACKETS,

/\* get/set green color dropped packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_GREEN\_DROPPED\_BYTES,

/\* get/set yellow color tx packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_ STAT\_YELLOW\_PACKETS,

/\* get/set yellow color tx bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_YELLOW\_BYTES,

/\* get/set yellow color drooped packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_YELLOW\_DROPPED\_PACKETS,

/\* get/set yellow color dropped bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_YELLOW\_DROPPED\_BYTES,

/\* get/set red color tx packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_RED\_PACKETS,

/\* get/set red color tx bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_RED\_BYTES,

/\* get/set red color dropped packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_RED\_DROPPED\_PACKETS,

/\* get/set red color drooped bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_RED\_DROPPED\_BYTES,

/\* get/set WRED green color dropped packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_GREEN\_DISCARD\_DROPPED\_PACKETS,

/\* get/set WRED green color dropped bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_GREEN\_DISCARD\_DROPPED\_BYTES,

/\* get/set WRED yellow color dropped packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_YELLOW\_DISCARD\_DROPPED\_PACKETS,

/\* get/set WRED yellow color dropped bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_YELLOW\_DISCARD\_DROPPED\_BYTES,

/\* get/set WRED red color dropped packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_RED\_DISCARD\_DROPPED\_PACKETS,

/\* get/set WRED red color dropped bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_RED\_DISCARD\_DROPPED\_BYTES,

/\* get/set WRED dropped packets count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_DISCARD\_DROPPED\_PACKETS,

/\* get/set WRED red dropped bytes count [uint64\_t] \*/

SAI\_QOS\_QUEUE\_STAT\_DISCARD\_DROPPED\_BYTES

} sai\_qos\_queue\_stat\_counter\_t;

/\*

\* Routine Description:

\* Get queue statistics counters.

\*

\* Arguments:

\* [in] queue\_id - Queue id

\* [in] counter\_ids - specifies the array of counter ids

\* [in] number\_of\_counters - number of counters in the array

\* [out] counters - array of resulting counter values.

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_qos\_queue\_stats\_fn)(

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

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

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

\_Out\_ uint64\_t\* counters

);

/\*

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

\*/

typedef struct \_sai\_qos\_api\_t

{

/\*\* QOS queue attributes \*/

sai\_set\_qos\_queue\_attribute\_fn set\_queue\_attribute;

sai\_get\_qos\_queue\_attribute\_fn get\_queue\_attribute;

sai\_get\_qos\_queue\_stats\_fn get\_queue\_stats;

} sai\_qos\_api\_t;

## Configuration examples

**Example assumes NPU supports the 8 unicast and 4 multicast h/w queues per port. How applications controls unicast queue and multicast queues.**

### Step-1: Get number of queue supported on port

int no\_of\_queues\_per\_port = 0;

sai\_object\_id\_t port\_id;

**sai\_object\_list\_t sai\_queue\_object\_list;**

sai\_attribute\_t sai\_attr;

sai\_attr.id = **SAI\_PORT\_ATTR\_QOS\_NUMBER\_OF\_QUEUES**;

sai\_attr.value.u32 = 0;

sai\_get\_port\_attribute\_fn(port\_id, 1, &sai\_attr);

no\_of\_queues\_per\_port = sai\_attr.value.u32;

### Step-2: Get list of SAI queue object id’s on port

sai\_attr.id = **SAI\_PORT\_ATTR\_QOS\_QUEUE\_LIST;**

sai\_attr.value.objlist.count = no\_of\_queues\_per\_port;

sai\_attr.value.objlist.list = calloc(no\_of\_queues\_per\_port, sizeof(sai\_object\_id\_t));

/\* Get queue list \*/

sai\_get\_port\_attribute\_fn(port\_id, 1, &sai\_attr);

sai\_queue\_object\_list.count = sai\_attr.value.objlist.count;

sai\_queue\_object\_list.list = calloc(sai\_queue\_object\_list.count, sizeof(sai\_object\_id\_t));

memcpy((char \*)sai\_queue\_object\_list.list, (char \*)sai\_attr.value.objlist.list, sizeof(sai\_object\_id\_t) \* sai\_queue\_object\_list.count);

### Step 3: Map Application queues to sai queues

Applications can expose CLI like unicast and multicast queues separately.

Applications needs to

|  |  |  |
| --- | --- | --- |
| Appl Q Type | Appl QID | SAI QID (sai\_object\_id\_t) |
| UCAST | 0 | sai\_queue\_object\_list.list[0] |
| UCAST | 1 | sai\_queue\_object\_list.list[1] |
| UCAST | 2 | sai\_queue\_object\_list.list[2] |
| UCAST | 3 | sai\_queue\_object\_list.list[3] |
| UCAST | 4 | sai\_queue\_object\_list.list[4] |
| UCAST | 5 | sai\_queue\_object\_list.list[5] |
| UCAST | 6 | sai\_queue\_object\_list.list[6] |
| UCAST | 7 | sai\_queue\_object\_list.list[7] |
| MCAST | 0 | sai\_queue\_object\_list.list[8] |
| MCAST | 1 | sai\_queue\_object\_list.list[9] |
| MCAST | 2 | sai\_queue\_object\_list.list[10] |
| MCAST | 3 | sai\_queue\_object\_list.list[11] |

int sai\_q\_type[12];

sai\_attribute\_t sai\_attr;

sai\_attr.id = SAI\_QOS\_QUEUE\_ATTR\_TYPE;

sai\_attr.value.s32 = 0;

int sai\_q = 0;

int no\_of\_ucast\_queues = 0;

int no\_of\_mcast\_queues = 0;

int no\_of\_queues = 0;

for (sai\_q = 0; sai\_q < sai\_queue\_object\_list.count; sai\_q++)

{

sai\_get\_qos\_queue\_attribute\_fn (sai\_queue\_object\_list.list[sai\_q], &sai\_attr);

sai\_q\_type[sai\_q] = sai\_attr.value.s32;

if(sai\_q\_type[sai\_q] == SAI\_QOS\_QUEUE\_TYPE\_UCAST)

no\_of\_ucast\_queues ++;

else if(sai\_q\_type[sai\_q] == SAI\_QOS\_QUEUE\_TYPE\_MCAST)

no\_of\_mcast\_queues ++;

else

no\_of\_queues ++;

}

sai\_object\_id\_t appl\_ucast\_qid\_to\_sai\_qId[8];

sai\_object\_id\_t appl\_mcast\_qid\_to\_sai\_qId[8];

sai\_object\_id\_t appl\_qid\_to\_sai\_qId[8];

int appl\_ucast\_q = 0;

int appl\_mcast\_q = 0;

int appl\_q = 0;

for (sai\_q = 0; sai\_q < sai\_queue\_object\_list.count; sai\_q++)

{

if(sai\_q\_type[sai\_q] == SAI\_QOS\_QUEUE\_TYPE\_UCAST)

{

appl\_ucast\_qid\_to\_sai\_qId [appl\_ucast\_q] = sai\_queue\_object\_list.list[sai\_q];

appl\_ucast\_q ++;

}

else if(sai\_q\_type[sai\_q] == SAI\_QOS\_QUEUE\_TYPE\_MCAST)

{

appl\_mcast\_qid\_to\_sai\_qId [appl\_mcast\_q] = sai\_queue\_object\_list.list[sai\_q];

appl\_mcast\_q ++;

}

else {

appl\_qid\_to\_sai\_qId [appl\_q] = sai\_queue\_object\_list.list[sai\_q];

appl\_q ++;

}

}

### Applying the wred to application unicast queue 0

sai\_attribute\_t sai\_queue\_attr\_set;

sai\_queue\_attr\_set.id = SAI\_QOS\_QUEUE\_ATTR\_DROP\_TYPE;

sai\_queue\_attr\_set.vlaue.s32 = SAI\_QOS\_DROP\_TYPE\_WRED ;

sai\_set\_qos\_queue\_attribute\_fn (appl\_ucast\_qid\_to\_sai\_qId[0], &sai\_queue\_attr\_set);

sai\_queue\_attr\_set.id = SAI\_QOS\_QUEUE\_ATTR\_WRED\_ID;

sai\_queue\_attr\_set.value.oid = wred\_id; // Created by saiwred object

sai\_set\_qos\_queue\_attribute\_fn (appl\_ucast\_qid\_to\_sai\_qId[0], &sai\_queue\_attr\_set);

### Applying the schedulers to Application multicast queue 3

sai\_attribute\_t sai\_attr;

sai\_attr.id.s32 = SAI\_QOS\_QUEUE\_ATTR\_SCHEDULER\_ID;

sai\_attr.value.oid = scheduler\_id\_1;

/\* Apply scheudler to multicast queue 3 \*/

sai\_set\_qos\_queue\_attribute\_fn (appl\_mcast\_qid\_to\_sai\_qId[3], sai\_attr);

### Map traffic class -> queues (8 queue model)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TC | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Queue | appl\_ucast\_qid\_to\_sai\_qId[0],  appl\_mcast\_qid\_to\_sai\_qId[0] | appl\_ucast\_qid\_to\_sai\_qId[1], appl\_mcast\_qid\_to\_sai\_qId[1] | appl\_ucast\_qid\_to\_sai\_qId[2], appl\_mcast\_qid\_to\_sai\_qId[2] | appl\_ucast\_qid\_to\_sai\_qId[3], appl\_mcast\_qid\_to\_sai\_qId[3] | appl\_ucast\_qid\_to\_sai\_qId[4], appl\_mcast\_qid\_to\_sai\_qId[3] | appl\_ucast\_qid\_to\_sai\_qId[5], appl\_mcast\_qid\_to\_sai\_qId[3] | appl\_ucast\_qid\_to\_sai\_qId[6], appl\_mcast\_qid\_to\_sai\_qId[3] | appl\_ucast\_qid\_to\_sai\_qId[7], appl\_mcast\_qid\_to\_sai\_qId[3] |

### Example Configurable Number of Queues per Port

Applications need only 4 unicast and 4 multicast in their platform. They can make the changes to qos traffic class -> queue change like below.

**Map traffic class -> queues(4 queue model)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TC | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Queue | appl\_ucast\_qid\_to\_sai\_**qId[0],** appl\_mcast\_qid\_to\_sai**\_qId[0]** | appl\_ucast\_qid\_to\_sai\_**qId[0],** appl\_mcast\_qid\_to\_sai\_**qId[0]** | appl\_ucast\_qid\_to\_sai\_**qId[1]**, appl\_mcast\_qid\_to\_sai\_**qId[1]** | appl\_ucast\_qid\_to\_sai\_qId[1], appl\_mcast\_qid\_to\_sai\_qId[1] | appl\_ucast\_qid\_to\_sai\_qId[2], appl\_mcast\_qid\_to\_sai\_qId[2] | appl\_ucast\_qid\_to\_sai\_qId[2], appl\_mcast\_qid\_to\_sai\_qId[2] | appl\_ucast\_qid\_to\_sai\_qId[3], appl\_mcast\_qid\_to\_sai\_qId[3] | appl\_ucast\_qid\_to\_sai\_qId[3], appl\_mcast\_qid\_to\_sai\_qId[3] |

# Appendix