

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **Buffers API’s** |
| **Authors** | **Mellanox** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **02/10/2015** |
| **SAI-Version** | **V0.9.4** |

**Contents**

[List of Changes i](#_Toc419926644)

[1 Overview 1](#_Toc419926645)

[1.1 Buffer pool 1](#_Toc419926646)

[1.2 Ingress/Egress buffer allocation model 1](#_Toc419926647)

[1.2.1 Ingress pipe 1](#_Toc419926648)

[1.2.2 Egress pipe 2](#_Toc419926649)

[1.2.3 Ingress and egress buffer admission rule 3](#_Toc419926650)

[1.2.4 XON/XOFF Frame generation model 3](#_Toc419926651)

[2 SAI model 3](#_Toc419926652)

[3 Specification 4](#_Toc419926653)

[3.1 New switch attributes 4](#_Toc419926654)

[3.2 New port attributes 4](#_Toc419926655)

[3.1 Ingress priority group (PG) configuration 5](#_Toc419926656)

[3.2 Buffer pool configuration 6](#_Toc419926657)

[3.2.1 Buffer profile configuration 8](#_Toc419926658)

[3.3 Buffer function Summary 9](#_Toc419926659)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.3 | Proposal for buffers |  | 2/10/15 |
| 0.9.3 | After internal review |  | 5/20/15 |
| 0.9.4 | Update ingress pipe buffer allocation model | Wenda Ni | 7/18/19 |

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

## Buffer pool

The switch buffer is divided into multiple buffer pools. A buffer pool is statically allocated buffer region. There are two types of buffer pool, ingress buffer pool and egress buffer pool. They are mapped the same switch buffer. An incoming packet must be admitted by an ingress buffer pool and an egress buffer pool.

## Ingress/Egress buffer allocation model

### Ingress pipe



### Egress pipe



### Ingress and egress buffer admission rule



### XON/XOFF Frame generation model

In lossless model, an ingress buffer have two extra attributes to control the generation of xoff/xon frames. When the available buffer in PG buffer is less than xoff threshold, xoff frame is triggered. When the total buffer usage for this PG (include both shared pool buffer, port buffer, and PG buffer usage) is less than xon threshold and available buffer is larger than xoff threshold, the xon frame is triggered.



# SAI model

Since in most case switch ports share the same buffer configuration, SAI supports creating buffer profiles and attaching those profiles to a port ingress PG, port egress queue. When the attributes in a buffer profile change, all attached buffers get affected.



# Specification

## New switch attributes

/\* READ-ONLY \*/

/\* switch total buffer size in KB [sai\_uint32\_t] \*/

SAI\_SWITCH\_ATTR\_TOTAL\_BUFFER\_SIZE,

/\* switch number of ingress buffer pool [sai\_uint32\_t] \*/

SAI\_SWITCH\_ATTR\_INGRESS\_BUFFER\_POOL\_NUM,

/\* switch number of egress buffer pool [sai\_uint32\_t] \*/

SAI\_SWITCH\_ATTR\_EGRESS\_BUFFER\_POOL\_NUM,

## New port attributes

/\* READ-ONLY \*/

/\* Priority group number [sai\_uint32\_t] \*/

SAI\_PORT\_ATTR\_PG\_NUM,

/\* list of ingress buffer ID [sai\_object\_list\_t]\*/

SAI\_PORT\_ATTR\_PG\_ID\_LIST,

/\* READ-WRITE \*/

/\* bit vector enable/disable port PFC [sai\_uint8\_t].

\* Valid from bit 0 to bit 7 \*/

SAI\_PORT\_ATTR\_PER\_PRIORITY\_FLOW\_CONTROL,

## Ingress priority group (PG) configuration

/\* buffer attribute \*/

typedef enum \_sai\_ingress\_priority\_group\_attr\_t

{

/\* buffer profile pointer [sai\_object\_id\_t] \*/

SAI\_INGRESS\_PRIORITY\_GROUP\_ATTR\_BUFFER\_PROFILE,

} sai\_ingress\_priority\_group\_attr\_t;

/\*

\* Routine Description:

\* Set ingress priority group attribute

\* Arguments:

\* [in] ingress\_pg\_id – ingress priority group id

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_ingress\_priority\_group\_attr\_fn)(

\_In\_ sai\_object\_id\_t ingress\_pg\_id,

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

);

/\*

\* Routine Description:

\* Get ingress priority group attributes

\* Arguments:

\* [in] ingress\_pg\_id – ingress priority group id

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

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

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_ingress\_priority\_group\_attr\_fn)(

\_In\_ sai\_object\_id\_t ingress\_pg\_id,

\_In\_ uint32\_t attr\_count,

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

);

## Buffer pool configuration

typedef enum \_sai\_buffer\_pool\_type\_t

{

/\* Ingress buffer pool \*/

SAI\_BUFFER\_POOL\_INGRESS,

/\* Egress buffer pool \*/

SAI\_BUFFER\_POOL\_EGRESS,

} sai\_buffer\_pool\_type\_t;

typedef enum \_sai\_buffer\_threshold\_mode\_t

{

/\* static maximum \*/

SAI\_BUFFER\_THRESHOLD\_MODE\_STATIC,

/\* dynamic maximum (relative) \*/

SAI\_BUFFER\_THRESHOLD\_MODE\_DYNAMIC,

} sai\_buffer\_threshold\_mode\_t;

/\* buffer attribute \*/

typedef enum \_sai\_buffer\_pool\_attr\_t

{

/\* READ ONLY shared buffer size in bytes [sai\_uint32\_t]. This is derived from substracting all reversed buffers of queue/port

From the total pool size. \*/

SAI\_BUFFER\_POOL\_ATTR\_SHARED\_SIZE,

/\* buffer pool type [sai\_buffer\_pool\_type\_t] (MANDATORY\_ON\_CREATE|CREATE\_ONLY) \*/

SAI\_BUFFER\_POOL\_ATTR\_TYPE,

/\* buffer pool size in bytes [sai\_uint32\_t] (MANDATORY\_ON\_CREATE|CREATE\_AND\_SET) \*/

SAI\_BUFFER\_POOL\_ATTR\_SIZE,

/\* shared threshold mode for the buffer pool [sai\_buffer\_threadhold\_mode\_t]

(CREATE\_AND\_SET default SAI\_BUFFER\_POOL\_DYNAMIC\_TH)

\*/

SAI\_BUFFER\_POOL\_ATTR\_TH\_MODE,

} sai\_buffer\_pool\_attr\_t;

/\*

\* Routine Description:

\* Create buffer pool

\*

\* Arguments:

\* [out] pool\_id -pool 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\_buffer\_pool\_fn)(

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

\_In\_ uint32\_t attr\_count,

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

);

/\*

\* Routine Description:

\* Remove buffer pool

\*

\* Arguments:

\* [in] pool\_id -pool id

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_buffer\_pool\_fn)(

\_In\_ sai\_object\_id\_t pool\_id

);

/\*

\* Routine Description:

\* Set buffer pool attribute

\* Arguments:

\* [in] pool\_id – pool id

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_buffer\_pool\_attr\_fn)(

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

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

);

/\*

\* Routine Description:

\* Get buffer pool attributes

\* Arguments:

\* [in] pool\_id – pool id

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

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

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_buffer\_pool\_attr\_fn)(

\_In\_ sai\_object\_id\_t\* pool\_id,

\_In\_ uint32\_t attr\_count,

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

);

### Buffer profile configuration

typedef enum \_sai\_buffer\_profile\_attr\_t

{

/\* pointer to buffer pool object [sai\_object\_id\_t] (MANDATORY\_ON\_CREATE|CREATE\_AND\_SET) \*/

SAI\_BUFFER\_PROFILE\_ATTR\_POOL\_ID,

/\* reserved buffer size in bytes [sai\_uint32\_t] (MANDATORY\_ON\_CREATE|CREATE\_AND\_SET) \*/

SAI\_BUFFER\_PROFILE\_ATTR\_BUFFER\_SIZE,

/\* dynamic threshold for the shared usage [sai\_uint8\_t]

The threshold is set to the 1/n of available buffer of the pool.

(CREATE\_AND\_SET).

Mandatory when SAI\_BUFFER\_POOL\_TH\_MODE = dynamic \*/

SAI\_BUFFER\_PROFILE\_ATTR\_SHARED\_DYNAMIC\_TH,

/\* static threshold for the shared usage in bytes [sai\_uint32\_t]

Mandatory when SAI\_BUFFER\_POOL\_TH\_MODE = static \*/

(CREATE\_AND\_SET)

SAI\_BUFFER\_PROFILE\_ATTR\_SHARED\_STATIC\_TH,

/\* set the buffer profile XOFF threshold in bytes [sai\_uint32\_t]

Valid only for ingress PG (CREATE\_AND\_SET).

Generate XOFF when available buffer in the PG buffer

is less than this threshold.

Default to SAI\_BUFFER\_PROFILE\_ATTR\_BUFFER\_SIZE.

\*/

SAI\_BUFFER\_PROFILE\_ATTR\_XOFF\_TH ,

/\* set the buffer profile XON threshold in byte [sai\_uint32\_t]

Valid only for ingress PG (CREATE\_AND\_SET)

Generate XON when the total buffer usage of this PG

is less this threshold and available buffer in the PG buffer

is larger than the XOFF threahold.

Default to 0.

\*/

SAI\_BUFFER\_PROFILE\_ATTR\_XON\_TH ,

}sai\_buffer\_profile\_attr\_t;

/\*

\* Routine Description:

\* Create buffer profile

\*

\* Arguments:

\* [Out] buffer\_profile\_id - buffer profile 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\_buffer\_profile\_fn)(

\_Out\_ sai\_object\_id\_t\* buffer\_profile\_id,

\_In\_ uint32\_t attr\_count,

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

);

/\*

\* Routine Description:

\* Remove buffer profile

\*

\* Arguments:

\* [in] buffer\_profile\_id - buffer profile id

\*

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_buffer\_profile\_fn)(

\_In\_ sai\_object\_id\_t buffer\_profile\_id

);

/\*

\* Routine Description:

\* Set buffer profile attribute

\* Arguments:

\* [in] buffer\_profile\_id - buffer profile id

\* [in] attr - buffer profile attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_buffer\_profile\_attr\_fn)(

\_In\_ sai\_object\_id\_t buffer\_profile\_id,

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

);

/\*

\* Routine Description:

\* Get buffer profile attributes

\* Arguments:

\* [in] buffer\_profile\_id - buffer profile id

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

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

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_buffer\_profile\_attr\_fn)(

\_In\_ sai\_object\_id\_t buffer\_profile\_id,

\_In\_ uint32\_t attr\_count,

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

);

## Buffer function Summary

/\* buffer method table retrieved with sai\_api\_query() \*/

typedef struct \_sai\_buffer\_api\_t

{

sai\_create\_buffer\_pool\_fn create\_buffer\_pool;

sai\_remove\_buffer\_pool\_fn remove\_buffer\_pool;

sai\_set\_buffer\_pool\_attr\_fn set\_buffer\_pool\_attr;

sai\_get\_buffer\_pool\_attr\_fn get\_buffer\_pool\_attr;

sai\_set\_ingress\_priority\_group\_attr\_fn set\_ingress\_priority\_group\_attr;

sai\_get\_ingress\_priority\_group\_attr\_fn get\_ingress\_priority\_group\_attr;

sai\_create\_buffer\_profile\_fn create\_buffer\_profile;

sai\_remove\_buffer\_profile\_fn remove\_buffer\_profile;

sai\_set\_buffer\_profile\_attr\_fn set\_buffer\_profile\_attr;

sai\_get\_buffer\_profile\_attr\_fn get\_buffer\_profile\_attr;

} sai\_buffer\_api\_t;