

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **Buffer Statistics Tracking (BST) API – A Broadcom Vendor Extension** |
| **Authors** | **Broadcom Limited** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **26-July-2016** |
| **SAI-Version** | **0.9.4** |

**Contents**

[List of Changes i](#_Toc457471981)

[1 Overview 1](#_Toc457471982)

[2 Proposal 1](#_Toc457471983)

[2.1 Statistics 1](#_Toc457471984)

[2.2 SAI bst Object 2](#_Toc457471985)

[2.3 Thresholds / Watermarks 2](#_Toc457471986)

[2.4 Tracking Mode 2](#_Toc457471987)

[2.5 Snapshots 2](#_Toc457471988)

[3 Specification 2](#_Toc457471989)

[3.1 New file saibst.h 2](#_Toc457471990)

[3.1.1 Data Structures & Enumerations 2](#_Toc457471991)

[3.1.2 API 4](#_Toc457471992)

[3.2 Changes to sai.h 7](#_Toc457471993)

[3.3 Changes to saitypes.h 7](#_Toc457471994)

[3.4 Changes to saiswitch.h 8](#_Toc457471995)

[3.5 Changes to saibuffer.h 8](#_Toc457471996)

[4 Examples 8](#_Toc457471997)

[4.1 Setting Up Buffer Tracking 8](#_Toc457471998)

[4.2 Creating and Reading from a Snapshot 9](#_Toc457471999)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 1 | Proposal for BST |  | 01/13/16 |
| 2 | Added description and adjusted the API |  | 02/24/16 |
| 3 | Added support for Bulk Statistic Retrieval & Thresholds |  | 05/25/16 |
| 4 | Minor changes to accommodate feedback |  | 06/12/16 |
| 5 | Support for CRUD Model |  | 06/25/16 |
| 6 | Reuse Existing SAI Objects |  | 06/26/16 |

License

© 2014 Microsoft Corporation, Dell Inc., Facebook, Inc, Broadcom Limited, 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

Networking Silicon sport a Memory Management Unit which manages packet buffers. The Silicon provides statistics to measure the buffer usage in the MMU. These statistics - Buffer Stats - allow user to get visibility into how the packet buffers are utilized.

This proposal defines a Public extension API for SAI, which allows users monitor the aforementioned buffer usage statistics. A hypothetical Networking silicon is described in terms of typical features and the API to monitor its statistics.

# Proposal

## Statistics

Networking Silicon reports buffer statistics for various on-chip buffers, part of the Memory Management Units. The buffers are presented under various categories, called *realms[[1]](#footnote-2)*. Thus, the buffer statistics are available on a per-realm basis.

The following table provides the list of realms for a typical Networking Silicon and the associated indices for each realm. It also lists all the available statistics for each realm.

|  |  |  |
| --- | --- | --- |
| Realm | Statistic Index | Statistic Name(s) |
| device |  | data |
| ingress-port-priority-group (port) | priority-group | um-share-buffer-count,  um-headroom-buffer-count |
| ingress-port-service-pool (port) | service-pool | um-share-buffer-count |
| ingress-service-pool | service-pool | um-share-buffer-count |
| egress-port-service-pool (port) | service-pool | uc-share-buffer-count,  um-share-buffer-count,  mc-share-buffer-count |
| egress-service-pool | service-pool | um-share-buffer-count,  mc-share-buffer-count |
| egress-uc-queue (port) | queue | uc-buffer-count |
| egress-uc-queue-group | queue-group | uc-buffer-count |
| egress-mc-queue (port) | queue | mc-buffer-count |
| egress-cpu-queue | queue | cpu-buffer-count |
| egress-rqe-queue |  | rqe-buffer-count |

Each buffer, given its parameters, may have one or more than one statistic. For example, the *Ingress Port-Service-Pool* realm buffer (for a given port), for a given service pool, offers a single statistic: um-share-buffer-count. The Eg*ress Port-Service-Pool* realm buffer (for a given port), for a given service pool, offers multiple statistics: *uc-share-buffer-count*, *um-share-buffer-count*, *mc-share-buffer-count*.

Much of the statistics are mapped to existing SAI objects and the statistics are retrieved by using the attribute retrieval function for the appropriate SAI object.

## SAI bst Object

A new object, called **bst**, is added as an attribute (**SAI\_SWITCH\_ATTR\_BST**) for the SAI switch object. This bst provides API to configure the buffer tracker on the silicon. It also provides API to carry out any bulk read/clear operations.

This object can be retrieved using the attribute retrieval function for the SAI switch object.

## Thresholds / Watermarks

Networking Silicon supports configuring watermark levels for each of the statistics. These watermarks are called *thresholds*. Once a threshold is configured for a statistic, the silicon can notify the CPU when the statistic value exceeds the threshold.

The thresholds can be configured on the silicon by using the attribute configuration function for the appropriate SAI object.

The API **sai\_bst\_threshold\_breach\_event\_notification\_fn** can be used to register a function to get notified of any threshold breach events.

## Tracking Mode

Networking Silicon allows tracking either the current values or the peak values of the statistics. By default, the current values for the statistics are tracked. The required mode can be configured with the **SAI\_BST\_BUFFER\_TRACKING\_MODE** attribute of the **bst** object, using the **sai\_bst\_attribute\_set\_fn** API.

## Snapshots

The Silicon supports taking a current snapshot of all the statistics, so that they can be used for a comparative analysis. A snapshot may be made by invoking the **sai\_bst\_snapshot\_create\_fn** API. This API returns a snapshot id, which can be used as an attribute for the **sai\_bst\_counters\_get** API to read all statistics from the snapshot.

Additionally, when a statistic values breaches the configured threshold for that statistic, then the driver automatically takes a snapshot of all the statistics and the snapshot id is provided as part of the event notification function.

# Specification

## New file saibst.h

### Data Structures & Enumerations

/\*\*

\* BST Statistic. Used at the time of bulk stat retrieval

\* This structure specifies both the statistic and its value.

\*/

typedef struct \_sai\_bst\_statistic\_t {

sai\_object\_id\_t port\_id;

sai\_object\_id\_t buf\_object\_id;

sai\_object\_type\_t object\_type;

uint32\_t statistic\_type;

uint64\_t value;

} sai\_bst\_statistic\_t;

typedef sai\_statistic\_t sai\_bst\_threshold\_t;

/\*\*

\* BST Statistics Reading Attributes.

\*/

typedef enum \_sai\_bst\_statistics\_attr\_t {

/\*\*

\* Snapshot to read counters from [sai\_bst\_snapshot\_id\_t] (OPTIONAL)

\* If not supplied, the data is read from the hardware and returned

\*/

SAI\_BST\_STATISTICS\_ATTR\_SNAPSHOT\_ID

} sai\_bst\_statistics\_attr\_t;

/\*\*

\* BST Tracking Options

\*/

typedef enum \_sai\_bst\_tracking\_options\_t

{

SAI\_BST\_TRACKING\_MODE\_PEAK,

SAI\_BST\_TRACKING\_MODE\_CURRENT

} sai\_bst\_tracking\_options\_t;

/\*\*

\* BST Reporting Options

\*/

typedef enum \_sai\_bst\_reporting\_options\_t

{

SAI\_BST\_REPORTING\_MODE\_BYTES,

SAI\_BST\_REPORTING\_MODE\_PERCENTAGE\_OF\_BUFFER\_SIZE,

} sai\_bst\_reporting\_options\_t;

/\*\*

\* BST Attributes.

\*/

typedef enum \_sai\_bst\_attr\_t {

/\* Number of counters supported [uint32\_t] (READONLY) \*/

SAI\_BST\_ATTR\_NUM\_STATISTICS,

/\* Operational State for the Buffer Tracking [boolean] (OPTIONAL) \*/

/\* Enabled by default \*/

SAI\_BST\_BUFFER\_TRACKING\_OPER\_STATE,

/\* Statistics reporting mode. [sai\_bst\_reporting\_options\_t] (OPTIONAL)\*/

/\* When not specified, reports in number of bytes (DEFAULT) \*/

SAI\_BST\_BUFFER\_REPORTING\_MODE,

/\*\*

\* Buffer Tracker Mode [sai\_bst\_tracking\_options\_t]. (OPTIONAL)

\* Specifies whether the Chip should track the peak values of the

\* buffers or current usage values (DEFAULT)

\*/

SAI\_BST\_BUFFER\_TRACKING\_MODE,

} sai\_bst\_attr\_t;

/\*\*

\* @brief BST Threshold Breach Event notification

\*/

typedef struct \_sai\_bst\_threshold\_breach\_event\_notification\_t {

/\*\* snapshot id \*/

sai\_object\_id\_t snapshot\_id;

/\*\* Statistic that caused the Threshold Breach event \*/

sai\_bst\_statistic\_t statistic;

} sai\_bst\_threshold\_breach\_event\_notification\_t;

### API

/\*\*

\* Routine Description:

\* @brief Create and return a snapshot object

\* This creates a snapshot in the hardware and copies the snapshot data

\* into the driver.

\*

\* Arguments:

\* @param[out] snapshot\_id - snapshot object

\* @param[in] bst\_id – bst object

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_bst\_snapshot\_create\_fn)(

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

\_In\_ sai\_object\_id\_t bst\_id

);

/\*\*

\* Routine Description:

\* @brief Return the object id for current snapshot maintained in the driver

\*

\* Arguments:

\* @param[out] snapshot\_id - snapshot object

\* @param[in] bst\_id – bst object

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_bst\_current\_snapshot\_get\_fn)(

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

\_In\_ sai\_object\_id\_t bst\_id

);

/\*\*

\* Routine Description:

\* @brief Deletes a specified snapshot object.

\*

\* Arguments:

\* @param[in] snapshot\_id - snapshot object to be removed.

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_bst\_snapshot\_remove\_fn)(

\_In\_ sai\_object\_id\_t \*snapshot\_id

);

/\*\*

\* Routine Description:

\* @brief Get values for specified BST attributes.

\*

\* Arguments:

\* @param[in] bst\_id – bst object id

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

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

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

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

\_In\_ uint32\_t attr\_count,

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

);

/\*\*

\* Routine Description:

\* @brief Set BST attribute value(s).

\*

\* Arguments:

\* @param[in] bst\_id - bst id

\* @param[in] attr - attribute

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

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

\_In\_ uint32\_t attr\_count,

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

);

/\*\*

\* Routine Description :

\* @brief Obtain the value for all statistics, either directly

\* from hardware or from a snapshot.

\*

\* Arguments:

\* @param[in] bst\_id – bst object id

\* @param[in] attr – specifies the optional snapshot id

\* @param[inout] stat\_count - number of statistics (required/provided)

\* @param[inout] statistics - statistics (allocated/provided)

\*

\* Return Values :

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_bst\_statistics\_get\_fn)(

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

\_In\_ uint32\_t attr\_count,

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

\_Inout\_ uint32\_t \*stat\_count,

\_Inout\_ sai\_bst\_statistic\_t \*statistics

);

/\*\*

\* Routine Description :

\* @brief Clear all the statistics

\*

\* Arguments:

\* @param[in] bst\_id – bst object id

\*

\* Return Values :

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_bst\_statistics\_clear\_fn)(

\_In\_ sai\_object\_id\_t bst\_id

);

/\*\*

\* Routine Description :

\* @brief Clear all the thresholds / watermarks

\*

\* Arguments:

\* @param[in] bst\_id – bst object id

\*

\* Return Values :

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_bst\_thresholds\_clear\_fn)(

\_In\_ sai\_object\_id\_t bst\_id

);

typedef struct \_sai\_bst\_api\_t

{

sai\_bst\_snapshot\_create\_fn bst\_snapshot\_create;

sai\_bst\_snapshot\_remove\_fn bst\_snapshot\_remove;

sai\_bst\_current\_snapshot\_get\_fn bst\_current\_snapshot\_get;

sai\_bst\_attribute\_get\_fn bst\_attribute\_get;

sai\_bst\_attribute\_set\_fn bst\_attribute\_set;

sai\_bst\_statistics\_get\_fn bst\_statistics\_get;

sai\_bst\_statistics\_clear\_fn bst\_statistics\_clear;

sai\_bst\_thresholds\_clear\_fn bst\_thresholds\_clear;

} sai\_bst\_api\_t;;

## Changes to sai.h

typedef enum \_sai\_api\_t {

…

SAI\_API\_BST = 0x200, /\* sai\_bst\_api\_t \*/

}

## Changes to saitypes.h

The following are the changes to the saitypes.h file.

* The union **sai\_object\_type\_t** is enhanced to include two new object types: **SAI\_OBJECT\_TYPE\_BST**, **SAI\_OBJECT\_TYPE\_BST\_SNAPSHOT**.

## Changes to saiswitch.h

The following are the changes to the saiswitch.h file

* The union **sai\_switch\_attr\_t** is enhanced to include three new attributes: **SAI\_SWITCH\_ATTR\_BST**, **SAI\_SWITCH\_ATTR\_BUFFER\_USAGE** and SAI**\_SWITCH\_ATTR\_BUFFER\_WATERMARK**.

## Changes to saibuffer.h

The following are the changes to the saibuffer.h file

* The enumeration **sai\_ingress\_priority\_group\_stat\_counter\_t** is enhanced to include three new counters: **SAI\_INGRESS\_PRIORITY\_GROUP\_STAT\_HEADROOM\_BYTES** and **SAI\_INGRESS\_PRIORITY\_GROUP\_STAT\_HEADROOM\_WATERMARK\_BYTES**.
* The enumeration **sai\_buffer\_pool\_stat\_counter\_t** is enhanced to include three new counters: **SAI\_BUFFER\_POOL\_STAT\_MULTICAST\_CURR\_OCCUPANCY\_BYTES** and **SAI\_BUFFER\_POOL\_MULTICAST\_STAT\_WATERMARK\_BYTES**.

# Examples

## Setting Up Buffer Tracking

sai\_object\_t bst;

sai\_attribute\_t switch\_attrs[1];

sai\_attribute\_t bst\_attrs[3];

/\* Get BST Object from switch \*/

switch\_attrs[0].id = (sai\_attr\_id\_t)SAI\_SWITCH\_ATTR\_BST;

get\_switch\_attribute(switch\_object\_id, 1, &switch\_attrs[0]);

bst = switch\_attrs[0].oid;

/\* Setup BST Global Configuration \*/

/\* Enable BST Tracking \*/

bst\_attrs[0].id = (sai\_attr\_id\_t)SAI\_BST\_BUFFER\_TRACKING\_OPER\_STATE;

bst\_attrs[0].value.booldata = 1;

/\* let the reporting be in percentages \*/

bst\_attrs[1].id = (sai\_attr\_id\_t)SAI\_BST\_BUFFER\_REPORTING\_MODE;

bst\_attrs[1].value.s32 = SAI\_BST\_REPORTING\_MODE\_PERCENTAGE\_OF\_BUFFER\_SIZE;

/\* we want peak values of buffer usage rather than current usage \*/

bst\_attrs[2].id = (sai\_attr\_id\_t)SAI\_BST\_BUFFER\_TRACKING\_MODE;

bst\_attrs[2].value.s32 = SAI\_BST\_TRACKING\_MODE\_PEAK;

/\* apply the configuration \*/

bst\_attribute\_set(bst, 3, &bst\_attrs[0]);

## Creating and Reading from a Snapshot

int num\_counters = 0;

sai\_bst\_statistic\_t \*stat\_list;

sai\_attribute\_t bst\_attrs[3];

sai\_object\_t snapshot;

sai\_object\_t bst;

/\* Obtain BST object \*/

...

/\* findout the number of available statistics \*/

bst\_attrs[0].id = (sai\_attr\_id\_t)SAI\_BST\_ATTR\_NUM\_STATISTICS;

bst\_attribute\_get(bst, 1, &bst\_attrs[0]);

num\_counters = bst\_attrs[0].value.s32;

/\* allocate memory for 'num\_counters' of 'sai\_bst\_statistic\_t' counters \*/

...

/\* create the snapshot \*/

bst\_snapshot\_create(&snapshot, bst);

/\* read from snapshot \*/

bst\_attrs[0].id = (sai\_attr\_id\_t)SAI\_BST\_STATISTICS\_ATTR\_SNAPSHOT\_ID;

bst\_attrs[0].value.oid = snapshot;

bst\_statistics\_get(bst, 1, &bst\_attrs[0], &num\_counters, stat\_list);

1. A hypothetical name [↑](#footnote-ref-2)