

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **SAI Multi NPU Part 1 (Switch ID)** |
| **Authors** | **Dell** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **8/5/2016** |
| **SAI-Version** | **1.0** |

**Contents**

[List of Changes i](#_Toc458115394)

[1 Overview 3](#_Toc458115395)

[2 Support local NPU addressability at the card/CPU level 4](#_Toc458115396)

[2.1 SAI object creation using SWITCH ID 4](#_Toc458115397)

[3 Specification 5](#_Toc458115398)

[3.1 Changes in saiswitch.h 5](#_Toc458115399)

[3.1.1 Notifications 9](#_Toc458115400)

[3.2 Adding switch Object ID to all Object’s 9](#_Toc458115401)

[3.3 Changes in saivlan.h 10](#_Toc458115402)

[3.4 Changes in saifdb.h 11](#_Toc458115403)

[4 Appendix A: Using threads or processes to identify local individual NPU’s/Switches 1](#_Toc458115404)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.3 | Base version |  | 7/8/2016 |
| 1.2.0 | Added switch OID, create/remove functions for switch object, switch ID as “mandatory at create” attribute |  | 8/5/2016 |

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

This proposal is intended to enhance the existing SAI API model in order to support multiple NPUs attached to a single CPU in a single-card system (multi-NPU pizza box)



CPU needs to address ports across all NPUs connected to it. A Switch ID can be used to distinguish port 1 on NPU x from port 1 on NPU y; essentially, each individual port is identified by the pair [Switch ID, Local NPU Port ID].

Use cases:

* LAG with members spanning multiple NPUs
* MAC learning & FDB management

Controlling SW executed on CPU’s that control multiple NPU’s need to be able to identify individual NPUs attached to/controlled from a given CPU.

This proposal adds a (local, to the CPU) “switch id” to SAI API in order to allow the adapter host to identify individual NPU’s.

The SAI API model is enhanced in order to allow the host-adaptor to control each NPU separately , i.e. SAI should be capable of performing initialization, de-initialization, restart and power-off each individual NPU.

It is assumed that the “switch ID” is a platform specific parameter, managed by the Host Adapter.

The rest of the document describes changes required to support local NPU addressability at the CPU level.

# Support local NPU addressability at the card/CPU level

Supporting individual NPU addressability requires the definition of a Switch ID. The Switch ID is used to identify individual NPU’s in all SAI functions (e.g. ‘create’ functions). The “switch ID” is local to a CPU.

* This provides SAI with capability to allow the host-adapter to address an individual NPU attached to a card/CPU.

## SAI object creation using SWITCH ID

Controlling multiple NPU from a single CPU can be achieved either by:

* Having separate SAI instances for each NPU in the system.
* A more detailed discussion about this approach is provided as an Appendix
* Controlling multiple NPU’s using a single SAI instance.

Note that this proposal “augments” the SAI API’s with a switch ID, thus allowing the host-adapter to use a single SAI instance to control multiple NPUs (vs. using the driver/SAI Instance approach).

Examples of operations that require addressing an individual NPU

* Switch init/deinit/restart/power-off per NPU.
* FDB entry can be programmed only in case VLAN has member ports in NPU.
* Multicast group id based on egress port list.

Current SAI specification defines:

**sai\_switch\_profile\_id\_t profile\_id:**

Host-adopter will pass profile Id, which will be used to retrieve values needed while initializing the switch . This profile id is used by “service\_method\_table” to get key-value pairs for this NPU.

* This approach will hold good even when multiple NPUs of same type are present in the same line card
* Scope of profile id is platform dependent.

**(SAI\_MAX\_HARDWARE\_ID\_LEN) char\* switch\_hardware\_id**

NPU details are represented as a string.

**Example:**

NPU Number, PCI Address location for each NPU.

**switch\_hardware\_id – NPUXXX-PCIXXX, NPUXXX-PCIYYY-1**.

# Specification

SAI object creation should be enhanced to support configurations at per-NPU granularity. Thus we propose to represent an NPU (physical switch) as switch SAI object and extend to it the CRUD model used for other SAI objects.

* A SAI “switch object” is created by using the hardware details (PCI location), firmware and profile. The create function returns a switch object ID.
* Use Switch Object ID’s in other objects to create any object at individual switch level. This attribute is valid only for create (CREATE\_ONLY) and also it is optional attribute for any object, if this attribute is not mentioned as part of creation it will use the default NULL Object ID.
  + The semantics of using the NULL object ID for switch ID is to represent the “first” (or unique) switch object created by an implementation. Thus, any implementation that only requires a single switch will not need to specify a switch ID attribute at object creation time.

Given below are changes needed to program each object at per-NPU granularity.

<https://github.com/opencomputeproject/SAI/compare/master...AshokDaparthi:switch-object>

## Changes in saiswitch.h

typedef char \*sai\_string\_t;

{

sai\_string\_t string;

} sai\_attribute\_value\_t;

typedef enum \_sai\_switch\_attr\_t

{

SAI\_SWITCH\_ATTR\_START,

/\*\* READ-ONLY \*/

/\*\* Handle for the switch profile.

\* sai\_switch\_profile\_id\_t of size \_In\_reads\_z\_(SAI\_MAX\_HARDWARE\_ID\_LEN) (MANDATORY\_ON\_CREATE|CREATE\_ONLY) \*/

SAI\_SWITCH\_ATTR\_PROFILE\_ID,

/\*\* Switch hardware ID to open

\* sai\_string\_t of size \_In\_reads\_z\_(SAI\_MAX\_HARDWARE\_ID\_LEN) (MANDATORY\_ON\_CREATE|CREATE\_ONLY) \*/

SAI\_SWITCH\_ATTR\_HARDWARE\_INFO,

/\*\* Vendor specific path name of the firmware to load

\* sai\_string\_t of size \_In\_reads\_z\_(SAI\_MAX\_FIRMWARE\_PATH\_NAME\_LEN) (MANDATORY\_ON\_CREATE|CREATE\_ONLY) \*/

SAI\_SWITCH\_ATTR\_FIRMWARE\_PATH\_NAME,

/\*\* Type of reboot WARM/COLD,

\* hint that indicates controlled warm restart.

\* Since warm restart can be caused by crash

\* (therefore there are no guarantees for this call),

\* this hint is really a performance optimization

\* [bool] Default to false(COLD)\*/

SAI\_SWITCH\_ATTR\_REBOOT\_WARM,

/\*\*

\* Routine Description:

\* Create Switch

\* - SDK initialization. After the call the capability attributes should be

\* ready for retrieval via sai\_get\_switch\_attribute().

\*

\* Arguments:

\* @param[out] switch\_id - router interface id

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

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

\* @param[in] switch\_notifications - switch notification table

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

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

\_In\_ uint32\_t attr\_count,

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

\_In\_ sai\_switch\_notification\_t\* switch\_notifications

);

/\*\*

\* Routine Description:

\* @brief Remove switch

\* - Release all resources associated with currently opened switch

)\*

\* Arguments:

\* @param[in] switch\_id - Switch id

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\*/

typedef void (\*sai\_remove\_switch\_fn)(

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

);

/\*\*

\* Routine Description:

\* SDK connect. This API connects library to the initialized SDK.

\* After the call the capability attributes should be ready for retrieval

\* via sai\_get\_switch\_attribute().

\*

\* Arguments:

\* @param[out] switch\_id - router interface id

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

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

\* @param[in] switch\_notifications - switch notification table

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_connect\_switch\_fn)(

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

\_In\_ uint32\_t attr\_count,

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

\_In\_ sai\_switch\_notification\_t\* switch\_notifications

);

/\*\*

\* Routine Description:

\* @brief Disconnect this SAI library from the SDK.

\*

\* Arguments:

\* None

\* Return Values:

\* None

\*/

typedef void (\*sai\_disconnect\_switch\_fn)(

\_In\_ sai\_object\_id\_t switch\_id

);

/\*\*

\* Routine Description:

\* @brief Set switch attribute value

\*

\* Arguments:

\* @param[in] switch\_id - Switch id

\* @param[in] attr - switch attribute

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

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

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

);

/\*\*

\* Routine Description:

\* @brief Get switch attribute value

\*

\* Arguments:

\* @param[in] switch\_id - Switch id

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

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

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

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

\_In\_ sai\_uint32\_t attr\_count,

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

);

/\*\*

\* Routine Description:

\* @brief Switch shutdown request callback.

\* Adapter DLL may request a shutdown due to an unrecoverable failure

\* or a maintenance operation

\*

\* Arguments:

\* None

\*

\* Return Values:

\* None

\*/

typedef void (\*sai\_switch\_shutdown\_request\_fn)(

\_In\_ sai\_object\_id\_t switch\_id

);

/\*\*

\* Routine Description:

\* @brief Switch oper state change notification

\*

\* Arguments:

\* @param[in] switch\_oper\_status - new switch oper state

\*

\* Return Values:

\* None

\*/

typedef void (\*sai\_switch\_state\_change\_notification\_fn)(

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

\_In\_ sai\_switch\_oper\_status\_t switch\_oper\_status

);

/\*\*

\* @brief Switch method table retrieved with sai\_api\_query()

\*/

typedef struct \_sai\_switch\_api\_t

{

sai\_create\_switch\_fn create\_switch;

sai\_remove\_switch\_fn remove\_switch;

sai\_connect\_switch\_fn connect\_switch;

sai\_disconnect\_switch\_fn disconnect\_switch;

sai\_set\_switch\_attribute\_fn set\_switch\_attribute;

sai\_get\_switch\_attribute\_fn get\_switch\_attribute;

} sai\_switch\_api\_t;

### Notifications

A switch ID parameter must be added to all notifications API’s (below), in order to identify the individual NPU which triggers a given notification.

The switch OID parameter is added to all notification functions

typedef struct \_sai\_switch\_notification\_t

{

sai\_switch\_state\_change\_notification\_fn on\_switch\_state\_change;

sai\_fdb\_event\_notification\_fn on\_fdb\_event;

sai\_port\_state\_change\_notification\_fn on\_port\_state\_change;

sai\_port\_event\_notification\_fn on\_port\_event;

sai\_switch\_shutdown\_request\_fn on\_switch\_shutdown\_request;

sai\_packet\_event\_notification\_fn on\_packet\_event;

} sai\_switch\_notification\_t;

For instance:

/\*\*

\* Routine Description:

\* @brief Switch oper state change notification

\*

\* Arguments:

**\* @param[in] switch\_id - switch identifier**

\* @param[in] switch\_oper\_status - new switch oper state

\*

\* Return Values:

\* None

\*/

typedef void (\*sai\_switch\_state\_change\_notification\_fn)(

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

\_In\_ sai\_switch\_oper\_status\_t switch\_oper\_status

);

## Adding switch Object ID to all Object’s

A switch object ID parameter is added to all objects, and can only be specified at create time. It shall also be possible to retrieve the Switch ID of any SAI object using get functions, with the corresponding switch ID attribute.

**Important Note**. The object ID returned by ‘create’ functions embeds the switch ID information. Thus, further set/get/remove operations do not need to specify the Switch ID attribute.

In addition, note the following assumptions:

- the SAI Object ID format is specific to a SAI implementation

- the SAI Object ID only local CPU and/or OS scope (an object ID cannot be transmitted/used across multiple CPUs, in a chassis or stack environment)

/\*\* Switch Object ID [sai\_object\_id\_t] (CREATE\_ONLY),

\* Default SAI\_NULL\_OBJECT\_ID means only in first switch/all swithes(TBD) \*/

SAI\_PORT\_ATTR\_SWITCH\_ID,

SAI\_VLAN\_ATTR\_SWITCH\_ID,

SAI\_LAG\_ATTR\_SWITCH\_ID,

SAI\_STP\_ATTR\_SWITCH\_ID,

SAI\_ACL\_TABLE\_ATTR SWITCH\_ID,

SAI\_ACL\_ENTRY\_ATTR\_SWITCH\_ID,

SAI\_ACL\_COUNTER\_ATTR\_SWITCH\_ID,

SAI\_ACL\_RANGE\_ATTR\_SWITCH\_ID,

SAI\_VIRTUAL\_ROUTER\_ATTR\_SWITCH\_ID,

SAI\_ROUTER\_INTERFACE\_ATTR\_SWITCH\_ID,

SAI\_NEXT\_HOP\_ATTR\_SWITCH\_ID

SAI\_NEXT\_HOP\_GROUP\_ATTR\_SWITCH\_ID

SAI\_NEIGHBOR\_ATTR\_SWITCH\_ID

SAI\_ROUTE\_ATTR\_SWITCH\_ID

SAI\_MIRROR\_SESSION\_ATTR\_SWITCH\_ID,

SAI\_SAMPLEPACKET\_ATTR\_SWITCH\_ID,

SAI\_QUEUE\_ATTR\_SWITCH\_ID,

SAI\_WRED\_ATTR\_SWITCH\_ID,

SAI\_POLICER\_ATTR\_SWITCH\_ID,

SAI\_NEXT\_HOP\_ATTR\_SWITCH\_ID,

SAI\_SCHEDULER\_GROUP\_ATTR\_SWITCH\_ID,

SAI\_HASH\_ATTR\_SWITCH\_ID,

## Changes in saivlan.h

typedef enum \_sai\_vlan\_attr\_t

{

/\*\* Switch Object ID [sai\_object\_id\_t] (CREATE\_ONLY),

\* Default SAI\_NULL\_OBJECT\_ID \*/

SAI\_VLAN\_ATTR\_SWITCH\_ID,

/\*\* vlan id [sai\_vlan\_id\_t] (MANDATORY\_ON\_CREATE|CREATE\_ONLY) \*/

SAI\_VLAN\_ATTR\_VLAN\_ID,

} sai\_vlan\_attr\_t;

/\*\*

\* Routine Description:

\* @brief Create a VLAN Object

\*

\* Arguments:

\* @param[out] vlan\_obj\_id - Vlan object id

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

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

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

\_Out\_ sai\_object\_id\_t\* vlan\_obj\_id,

\_In\_ uint32\_t attr\_count,

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

);

/\*\*

\* Routine Description:

\* @brief Remove a VLAN

\*

\* Arguments:

\* @param[in] vlan\_obj\_id - VLAN id

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

\_In\_ sai\_object\_id\_t vlan\_obj\_id

);

/\*\*

\* Routine Description:

\* @brief Set VLAN attribute Value

\*

\* Arguments:

\* @param[in] vlan\_id - VLAN id

\* @param[in] attr - attribute

\*

\* Return Values:

\* @return SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

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

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

);

/\*\*

\* Routine Description:

\* @brief Get VLAN attribute Value

\*

\* Arguments:

\* @param[in] vlan\_id - VLAN 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\_get\_vlan\_attribute\_fn)(

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

\_In\_ uint32\_t attr\_count,

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

);

## Changes in saifdb.h

typedef struct \_sai\_fdb\_entry\_t

{

sai\_mac\_t mac\_address;

sai\_vlan\_id\_t vlan\_id;

**sai\_object\_id\_t switch\_id;**

} sai\_fdb\_entry\_t;

# Appendix A: Other Approaches - Using threads or processes to identify local individual NPU’s/Switches

Theoretically, it would be possible to start a separate process (or thread) for each individual NPU. While this avoids pervasive changes to the existing SAI API, this approach simply “pushes” the problem to a higher application layer and implies a certain process architecture and assumption about the SAI implementation (and thus is not preferred). Any multi-NPU aware host-adapter or application needs to:

- Manage a table of mappings of process/thread ID to switch ID

- (Likely) Implement switch ID aware “wrapper” functions

Appendix B

OID unique across NPU’s (example: VLAN Id, switch ) -> unique OID

Get/set do not need switch ID

get\_switch\_oid(any OID) 🡪 Error is NULL OID