



# OPEN

Compute Project

## Switch Abstraction Interface Change Proposal

<b>Title</b>	SAI Multi NPU
<b>Authors</b>	Dell
<b>Status</b>	In Review
<b>Type</b>	Standards Track
<b>Created</b>	7/8/2016
<b>SAI-Version</b>	0.9.3

## Contents

List of Changes .....	i
1 Overview .....	1
2 SAI object creation by NPU'ID.....	<u>332</u>
2.1 Specification.....	<u>343</u>
2.1.1 Changes in saiswitch.h .....	<u>343</u>
2.1.2 Changes in saipor.h.....	<u>565</u>
2.1.3 Changes in saivlan.h.....	<u>665</u>
2.1.4 Changes in saifdb.h .....	<u>887</u>
2.1.5 Changes in sailag.h.....	8
2.1.6 Changes in saistp.h.....	8
2.1.7 Changes in saiacl.h .....	<u>998</u>
2.1.8 Changes in Routing .....	9
2.1.9 Changes in mirror.....	9
2.1.10 Changes in sflow .....	9
2.1.11 Changes in QOs .....	<u>9109</u>
3 SAI Unique Fabric/Module ID per NPU .....	<u>10109</u>
3.1 Specification.....	10
3.1.1 Changes in saiswitch.h .....	10
4 SAI Remote/Global OID.....	<u>101110</u>
4.1 Unique OID:.....	<u>111110</u>
4.2 Exchange SAI OID's between SAI instances. ....	12
4.3 Specification.....	12
4.3.1 Changes in saipor.h.....	12
5 SAI Port as Fabric Port.....	13
5.1 Specification.....	<u>131413</u>
6 SAI Fabric Route Table .....	<u>151513</u>
6.1 Specification.....	<u>151514</u>
7 SAI Object Creation By passing ID .....	14
➤ SAI LAG object.....	14

|

➤	SAI VxLAN Tunnel.....	14
➤	SAI Multicast ID.....	<u>1415</u>
7.1	Specification.....	15

## List of Changes

Version	Changes	Name	Date
0.9.3	Base version		7/8/2016

## 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.

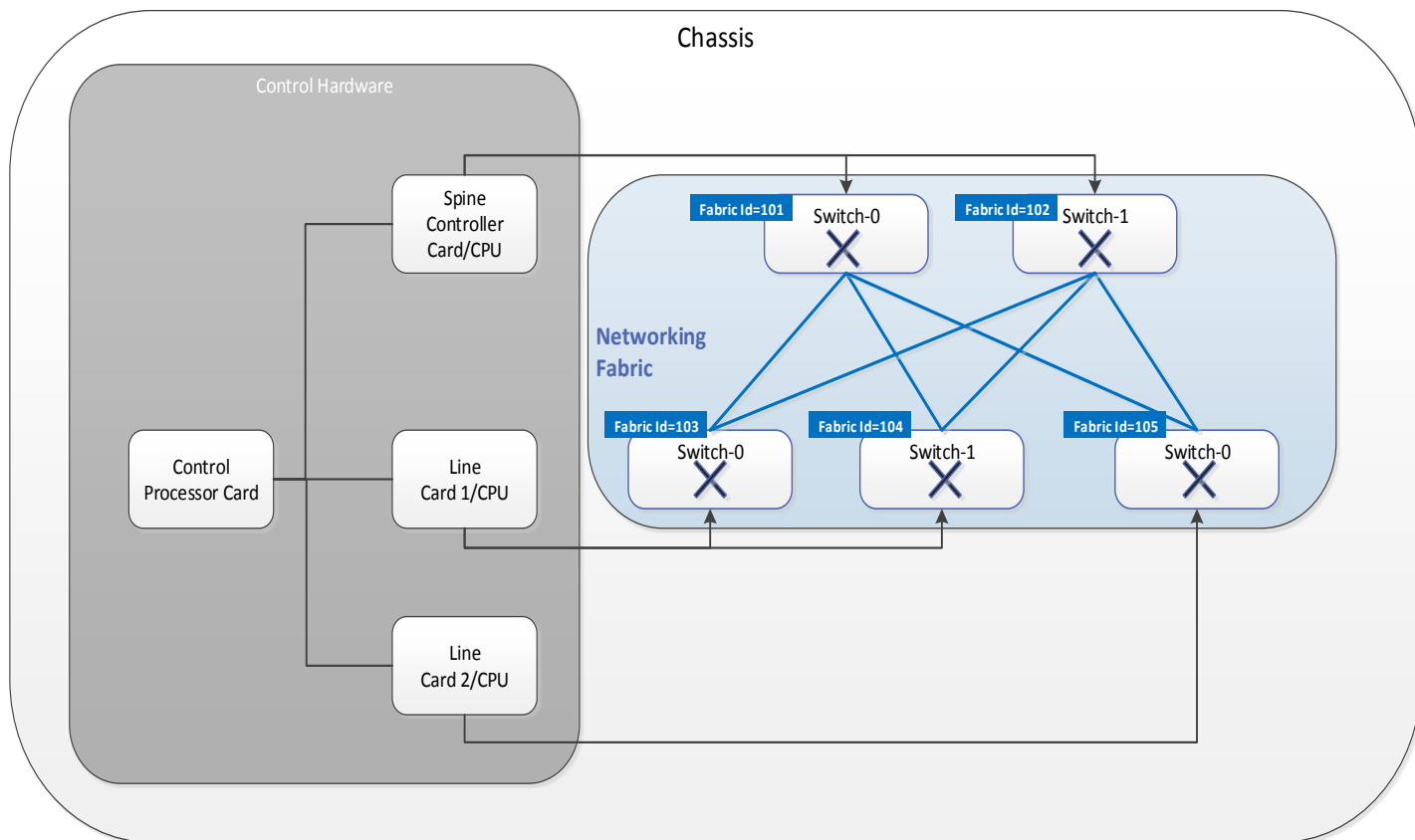
Your use of this Specification may be subject to other third party rights. THIS SPECIFICATION IS PROVIDED "AS IS." The contributors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, noninfringement, fitness for a particular purpose, or 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.

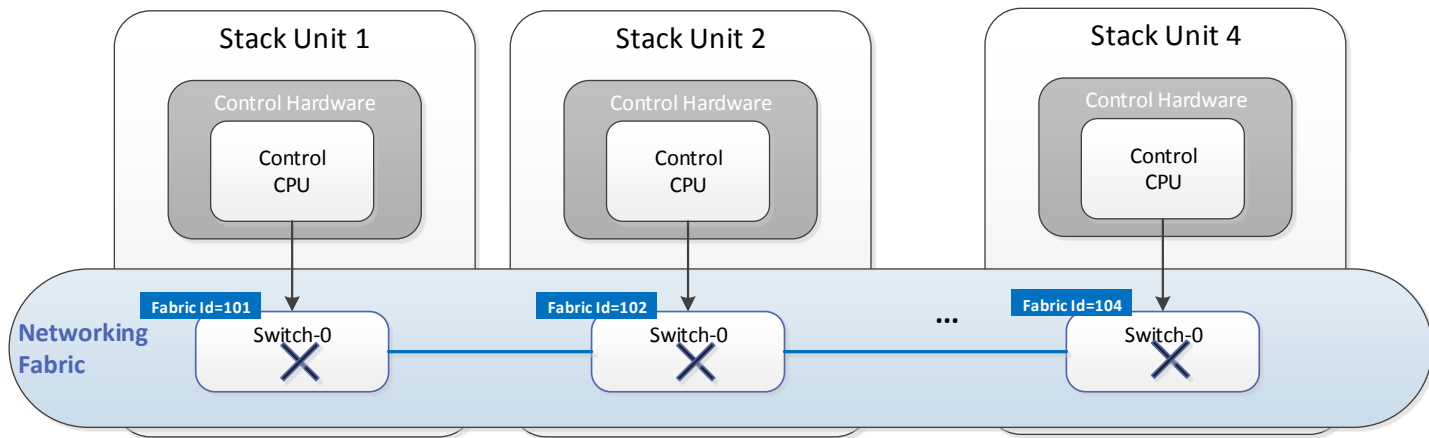
## 1 Overview

This proposal is intended to enhance the existing SAI API model in order to support multiple NPU's, either on multi-card or single-card systems, such as:

- chassis with control processors (CP) and linecards(LC)
- stacking
- multiple NPU's attached to and controlled from a single card/CPU system (multi-NPU 'pizza-boxes')



**Chassis System – with a Spine-Leaf networking fabric**



## Stacking System

The figures above illustrate two important concepts related to multi-NPU systems:

- Controlling SW executed on CPU's that control multiple NPU's (e.g. "Line Card 1" and "Spine Controller", in the figure above) need to be able to identify individual NPUs attached to/controlled from a given CPU.
  - o This proposal adds a (local, to the CPU) "switch id" to SAI API in order to allow the host-adaptor 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 initialize, de-init, restart and power-off each individual NPU and so forth.
- The networking fabric needs separate global switch identifiers ("fabric IDs") associated to each NPU in a networking fabric, whether spine-leaf or stacking (or other topologies)
  - o For instance, the fabric ID can be used to distinguish port 1 on NPU x from port 1 on NPU y; essentially, each individual port is globally identified by the pair [ Global Switch ID/Fabric ID, Local NPU Port ID ]; use cases:
    - MLAG / multi-NPU LAG
    - Chassis wide or stack wide MAC address discovery/FDB management

Note. It is assumed that both the "switch ID" and "fabric ID" are platform specific parameters, managed by the Host Adapter. In fact, determining "fabric ID's" may be the responsibility of higher software layers, but this is outside the scope of the SAI API specification.

The following major items should be considered in order to allow the SAI API to provide full multi-NPU support:

1# Defining a Switch ID in order to identify individual NPU's in SAI functions (e.g. 'create' functions). The "switch ID" is local to a CPU.

2# Assigning a unique ID to each NPU in a networking fabric (Fabric/module ID).

3# Remote/Global ports (global port identifier in a networking fabric)

4# SAI object creation by passing ID (WITHID) –needed for LAG and VxLAN

- In a multi-NPU networking fabric, multi-NPU LAG / trunk groups must be uniquely identified by a global LAG ID; same concept applies to other SAI objects

5# Fabric route table management.

## 2 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.
- 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 will pass the profile Id, this will be used to retrieve values needed switch init by using service\_method table.
- This can be same incase if we have the similar NPU in line card.
- Scope of this platform dependent.

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

This is to represent switch NPU details as string.

**Example:**

NPU number, PCI location and driver instance or only the driver instance to use for particular NPU.

**switch\_hardware\_id – NPUXXX-PCIXXX-0 , NPUXXX-PCIYYY-1 or 0, 1**

### 2.1 Specification

SAI object creation should be enhanced to supported configuration per NPU granularity. 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 SWITCH id i.e 0.

#### 2.1.1 Changes in saiswitch.h



```

typedef UINT32 sai_switch_id_t;
/**
 * Routine Description:
 * SDK initialization. After the call the capability attributes should be
 * ready for retrieval via sai_get_switch_attribute().
 *
 * Arguments:
 * @param[in] profile_id - Handle for the switch profile.
 * @param[in] switch_hardware_id - Switch hardware ID to open
 * @param[in] firmware_path_name - Vendor specific path name of the firmware
 *                               to load
 * @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_initialize_switch_fn)(
    _In_ sai_switch_profile_id_t profile_id,
    _In_ sai_switch_id_t switch_id,
    _In_reads_z_(SAI_MAX_HARDWARE_ID_LEN) char* switch_hardware_id,
    _In_reads_opt_z_(SAI_MAX_FIRMWARE_PATH_NAME_LEN) char* firmware_path_name,
    _In_ sai_switch_notification_t* switch_notifications,
);

/**
 * Routine Description:
 * @brief Release all resources associated with currently opened switch
 *
 * Arguments:
 * @param[in] warm_restart_hint - 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.
 *
 * Return Values:
 * None
 */
typedef void (*sai_shutdown_switch_fn)(
    _In_ sai_switch_id_t switch_id,
    _In_ bool warm_restart_hint
);

/**
 * Routine Description:
 * @brief Disconnect this SAI library from the SDK.
 *
 * Arguments:
 * None
 * Return Values:
 * None
 */
typedef void (*sai_disconnect_switch_fn)(
    _In_ sai_switch_id_t switch_id
);

/**
 * Routine Description:
 * @brief Set switch attribute value
 *

```

```

* Arguments:
*   @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_switch_id_t switch_id,
    _In_ const sai_attribute_t *attr
);

/**
* Routine Description:
*   @brief Get switch attribute value
*
* Arguments:
*   @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_switch_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_switch_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)( sai_switch_id_t switch_id,
    _In_ sai_switch_oper_status_t switch_oper_status
);

```

## 2.1.2 Changes in saiport.h

```
typedef enum _sai_port_attr_t
{
    /** READ-ONLY */
    SAI_PORT_ATTR_SWICH_ID,
} sai_port_attr_t;
```

### 2.1.3 Changes in saivlan.h

```
typedef enum _sai_vlan_attr_t
{
    /** READ-ONLY */
    SAI_VLAN_ATTR_SWITCH_ID,
} sai_vlan_attr_t;

/**
 * Routine Description:
 * @brief Create a VLAN
 *
 * Arguments:
 * @param[in] vlan_id - VLAN id
 *
 * Return Values:
 * @return SAI_STATUS_SUCCESS on success
 *         Failure status code on error
 */
typedef sai_status_t (*sai_create_vlan_fn)(
    _In_ sai_switch_id_t switch_id;
    _In_ sai_vlan_id_t vlan_id
);

/**
 * Routine Description:
 * @brief Remove a VLAN
 *
 * Arguments:
 * @param[in] vlan_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_switch_id_t switch_id;
    _In_ sai_vlan_id_t vlan_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_switch_id_t switch_id;
    _In_ sai_vlan_id_t vlan_id,
```

```

    _In_ const sai_attribute_t *attr
);

/**
 * Routine Description:
 *   @brief Remove VLAN configuration (remove all VLANs).
 *
 * Arguments:
 *   None
 *
 * Return Values:
 *   @return SAI_STATUS_SUCCESS on success
 *           Failure status code on error
 */
typedef sai_status_t (*sai_remove_all_vlans_fn)(
    _In_ sai_switch_id_t switch_id
);

/**
 * Routine Description:
 *   @brief Add Port to VLAN
 *
 * Arguments:
 *   @param[in] vlan_id - VLAN id
 *   @param[in] port_count - number of ports
 *   @param[in] port_list - pointer to membership structures
 *
 * Return Values:
 *   @return SAI_STATUS_SUCCESS on success
 *           Failure status code on error
 */
typedef sai_status_t (*sai_add_ports_to_vlan_fn)(
    _In_ sai_switch_id_t switch_id,
    _In_ sai_vlan_id_t vlan_id,
    _In_ uint32_t port_count,
    _In_ const sai_vlan_port_t *port_list → This is switch local port list.
);

Or
Based on port oid, we can configure it on specific NPU internally.

/**
 * Routine Description:
 *   @brief Get vlan statistics counters.
 *
 * Arguments:
 *   @param[in] vlan_id - VLAN id
 *   @param[in] counter_ids - specifies the array of counter ids
 *   @param[in] number_of_counters - number of counters in the array
 *   @param[out] counters - array of resulting counter values.
 *
 * Return Values:
 *   @return SAI_STATUS_SUCCESS on success
 *           Failure status code on error
 */
typedef sai_status_t (*sai_get_vlan_stats_fn)(
    _In_ sai_switch_id_t switch_id,
    _In_ sai_vlan_id_t vlan_id,
    _In_ const sai_vlan_stat_counter_t *counter_ids,
    _In_ uint32_t number_of_counters,
    _Out_ uint64_t* counters
);

```

### 2.1.4 Changes in saifdb.h

```
typedef enum _sai_fdb_entry_attr_t
{
    /** FDB entry on NPU (CREATE_ONLY),
        Default to 0 */
    SAI_FDB_ENTRY_ATTR_SWITCH_ID,
}

/**
 * Routine Description:
 *     @brief Remove FDB entry
 *
 * Arguments:
 *     @param[in] fdb_entry - fdb entry
 *
 * Return Values:
 *     @return SAI_STATUS_SUCCESS on success
 *     Failure status code on error
 */
typedef sai_status_t (*sai_remove_fdb_entry_fn)(
    _In_ sai_switch_id_t switch_id,
    _In_ const sai_fdb_entry_t* fdb_entry
);

Or

typedef struct _sai_fdb_entry_t
{
    sai_mac_t mac_address;
    sai_vlan_id_t vlan_id;
    sai_switch_id_t switch_id;
} sai_fdb_entry_t;
```

### 2.1.5 Changes in sailag.h

```
typedef enum _sai_lag_attr_t
{
    /** CREATE_ONLY),
        Default to 0 */
    SAI_LAG_ATTR_SWITCH_ID,
} sai_lag_attr_t;
```

### 2.1.6 Changes in saistp.h

```
typedef enum _sai_stp_attr_t
{
    /** CREATE_ONLY),
        Default to 0 */
    SAI_STP_ATTR_SWITCH_ID,

    /** READ-WRITE */
} sai_stp_attr_t;
```

### 2.1.7 Changes in saiacl.h

```
typedef enum _sai_acl_table_attr_t
{
    /** CREATE_ONLY),
        Default to 0 */
    SAI_ACL_TABLE_ATTR_SWITCH_ID,

    /** READ-WRITE */
} sai_acl_table_attr_t;

typedef enum _sai_acl_entry_attr_t
{
    /** CREATE_ONLY),
        Default to 0 */
    SAI_ACL_ENTRY_ATTR_SWITCH_ID,

    /** READ-WRITE */
} sai_acl_table_attr_t;

    /** CREATE_ONLY),
        Default to 0 */
    SAI_ACL_COUNTER_ATTR_SWITCH_ID,

    /** CREATE_ONLY),
        Default to 0 */
    SAI_ACL_RANGE_ATTR_SWITCH_ID,
```

### 2.1.8 Changes in Routing

```
    /** CREATE_ONLY),
        Default to 0 */
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
```

### 2.1.9 Changes in saimirr.h

```
    /** CREATE_ONLY),
        Default to 0 */
SAI_MIRROR_SESSION_ATTR_SWITCH_ID,
```

### 2.1.10 Changes in saisamplepacket.h

```
    /** CREATE_ONLY),
        Default to 0 */
SAI_SAMPLEPACKET_ATTR_SWITCH_ID,
```

### 2.1.11 Changes in QOs

```
    /** CREATE_ONLY),
```

```

        Default to 0 */
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,

```

### 3 SAI Unique Fabric/Module ID per NPU

In Chassis or stacking systems each NPU in system needs the unique identifier for fast path switching. By looking at this identifier packets are switched to destination NPU.

This is not NPU ID, NPU ID scope is local to CPU. This Fabric or module ID scope is global to system.

#### Topology Discovery:

General chassis or stacking model will have their own topology discovery modules to find how many NPU in the system and how they are connected each other. After discovering topology unique fabric/module ID will be assigned to each NPU in the system.

#### Dynamic Discovery:

- Every unit will participate in discovery with existing fabric id. Example Default fabric id 0 for all units.
- Once Topology discovered, each unit will be assigned with correct unique fabric ID in that system.

#### Fixed Topology:

- In Fixed Chassis, User can predefine fabric id's for each NPU in their product.

SAI should have option to set the fabric id as part of switch initialization or after switch init. This depends on the NPU capability.

#### 3.1 Specification

[This section describes an overview of the proposed interface/API.]

##### 3.1.1 Changes in saiswitch.h

```

/** READ-WRITE */

/* Set Fabric ID allowed only after deleting all SAI objects from switch*/
SAI_SWITCH_ATTR_FABRIC/MODULE_ID

#define SAI_KEY_SWITCH_FABRIC_ID "SAI_SWITCH_FABRIC_ID"

```

### 4 SAI Remote/Global OID

In a Multi-NPU system, global or system-wide NPU ports refer to the union of all ports on all NPUs on all line cards or stack members.

Each NPU has its own SAI instance. Each SAI instance will allocate SAI Port ID for ports on its own NPU. But since these port IDs are local to each NPU, the Port IDs generated by two different SAI Instances will overlap.

In the fast-path packet processing pipeline, the MAC and Route Table entries are looked up at the ingress NPU whereas the destination port may be on a different destination NPU. Since looking up the MAC or Route table in both the ingress and destination NPUs affects throughput, typically NPUs use the concept of Global NPU ports to refer to remote ports - so the MAC and Route table entries at the Ingress NPU are programmed with Global NPU Port ID of the remote NPU port.

In order to support this every SAI instance participating in a multi-NPU system should also support the concept of global NPU ports.

#### 4.1 Unique OID:

MAC or Route table can be pointed to ports in the other NPU, internally H/W tables directly programmed with remote port or fabric + port combination. Based on this packets are tx to remote NPU.

SAI OID representation is internal to each SAI vendor implementation. In case more than one SAI implementation for same NPU vendor and this combination we are using it same chassis. Then remote port representation by one implementation should be known to other unless we have explicit pass through mechanism like port and fabric id per port from host.

##### Approach 1: Fixed OID format

To make SAI oid's are unique per each NPU in topology, we can include Fabric ID as part of OID.

To support this we can define the SAI OID format so that every vendor will use this format for create OID. By doing this Any SAI implementation can derive the Fabric ID from the OID itself.

##### Example: Port UOID Representation:

SAI UOID includes,

Object Type : Type of the object : SAI\_OBJECT\_PORT

Object Id : Id corresponding to the object Type : SAI PORT ID

Reserved : Reserved field for future

Reserved	Objects Type	Object Id
(4)	(12)	(48)

##### SAI Port Object ID Representation:



A SAI port (UINT64) can have the following information embedded in it

- Port Type - Supports more than just Front-Panel user Port or CPU port.
- NPU/Fabric ID - Needed in a system with Multi NPU's, we need to program a distinct Fabric ID for each NPU in the system
- NPU Logical Port Number. Logical Port numbers can be same as HW Port number/gport or it can be mapped to HW port depending on the NPU..

Reserved	Port Type	Fabric Id	Port Number
4 Bits	4 Bits	8 bits	32 Bits
47 – 44	43 – 40	39 – 32	31 – 0

### Approach 2: Configure Fabric ID per port.

For local SAI port oid's, Fabric ID can be derived from Switch attribute or you can set this as part of create/set port attributes.

#### 4.2 Exchange SAI OID's between SAI instances.

SAI port OID format not generic, To use remote port OID in the local SAI instance, host adaptor has to pass oid's created in one SAI instance to other units. Remote port OID's are cached in the other SAI instances other than the local SAI. This can be used for validations and configure the fabric ID for remote port OID.

This can be avoided incase if we fix the OID format, so that SAI implementation will understand remote port OID and just it for getting Fabric ID. Then we no need to exchange any information of remote OID's. While Creating we have to take care SAI OID is created with correct Fabric ID included.

#### 4.3 Specification

This section describes an overview of the proposed interface/API for approach 2.

##### 4.3.1 Changes in saiport.h

```
/* Local port oid, Default - SAI_SWITCH_ATTR_FABRIC_ID */
SAI_PORT_ATTR_FABRIC/MIDULE_ID
```

```
/** Local Ports Creation for NPU */
typedef sai_status_t(*sai_create_port_fn)(
    _Out_ sai_object_id_t* port_id,
    _In_ uint32_t attr_count,
    _In_ sai_attribute_t *attr_list
);
```

```
/** Local Ports remove for NPU */
```

```

typedef sai_status_t(*sai_remove_port_fn)( _In_ sai_object_id_t port_id)

/**
 * Routine Description:
 *   @brief Add remote ports to a SAI.
 *
 * Arguments:
 *   @param[in] port_count - number of ports
 *   @param[in] port_oid - array of remote port oid's
 *
 * Return Values:
 *   @return SAI_STATUS_SUCCESS on success
 *           Failure status code on error
 */
typedef sai_status_t (*sai_add_remote_ports_fn)(
    _In_ uint32_t port_count,
    _In_ const sai_object_id_t* port_oids
);

/**
 * Routine Description:
 *   @brief Remove remote ports oids from a SAI
 *
 * Arguments:
 *   @param[in] port_count - number of ports
 *   @param[in] port_oids - array of remmote port oids.
 *
 * Return Values:
 *   @return SAI_STATUS_SUCCESS on success
 *           Failure status code on error
 */
typedef sai_status_t (*sai_remove_remote_ports_fn)(
    _In_ uint32_t port_count,
    _In_ const sai_object_id_t* port_oids
);

typedef struct _sai_port_api_t
{
    sai_add_remote_ports_fn      add_ports;
    sai_remove_remote_ports_fn   remove_ports;
} sai_port_api_t;

```

## 5 SAI Port as Fabric Port

In Chassis or staking some port in in NPU will be used to connect as fabric links or backplane. This port may carry the packet from one unit to other unit with special encapsulation etc.

### 5.1 Specification

```

/**
 * @brief Attribute data for SAI_PORT_ATTR_TYPE
 */
typedef enum _sai_port_encp_type_t
{

```

```

    /** Actual port. N.B. Different from the physical port. */
    SAI_PORT_ENCAP_TYPE_ETHERNET,

    /** CPU Port */
    SAI_PORT_ENCAP_TYPE_FABRIC,
} sai_port_encap_type_t;

/* Default - SAI_PORT_ENCAP_TYPE_ETHERNET*/

SAI_PORT_ATTR_ENCAP_TYPE;

```

## 6 SAI Object Creation By passing ID

There is a requirement to have the notion of global Port OID as captured in the Section 4 SAI Remote/Global OID. Further, there is also a need for having global OID for certain SAI objects in the Multi NPU system. This is required when the forwarding lookup happens only in ingress NPU and the destination information is carried as meta-data to the egress NPU. Some examples are captured below.

### Examples:

#### ➤ SAI LAG object

- SAI LAG object can have member ports spanning across multiple NPU in the system. If a packet enters the ingress NPU on one of the LAG member ports, the Source LAG Id assigned for the packet is usually carried as a meta-data to the remote NPU for reasons like flooding the packets to member ports in the remote NPU.
- This requires the LAG Id to be globally unique across the multiple NPUs.
- 

#### ➤ SAI VxLAN Tunnel

- SAI VxLAN maps the bridge port to a VNI and FDB entry on the access network is learnt on the bridge port
- Bridge port Id has to be globally unique across the multiple NPUs just like the port OID so that the FDB entry can point to a bridge port id in the remote NPU.
- 

#### ➤ SAI Multicast Group ID

- When a packet is destined to a Multicast Group Id that has member ports in remote NPU, the Multicast Group Id has to be looked up in the remote NPU for replicating to the local egress ports that are part of the group.
- This requires the multicast group id to be globally unique across the multiple NPUs.
- 

To provide the more flexibility and control to host, SAI API needs to be extended to program object with the given Id. This will also solve the Multi NPU requirements.

## 6.1 Specification

Proposal is to have the attribute in all the objects as platform specific and opaque (to the host adapter) “hardware identifier”. This identifier needs to be provided at object creation time.

**Example:**

```
/** SAI LAG HWID list [sai_u32_list_t] */  
SAI_LAG_ATTR_HWID_LIST,
```

## 7 SAI Fabric Route Table

In general, in a Chassis or Stacking environment, routing traffic from one NPU to other NPU is managed by a routing table based on fabric id’s and fabric ports.

This table helps to:

- Obtain the shortest path to reach the destination NPU.
- Preventing loops.

This table can be changed based on topology changes and this topology changes can be detected from software or hardware. Controlling this table form host depends on NPU/SDK capability.

### 7.1 Specification

```
/** Fabric ID list [uint8_t]  
 * Port can be used to route traffic to fabric in the list.  
 * Modify needs to send the new list.  
 */  
SAI_PORT_ATTR_ROUTE_FABRIC_ID_LIST,
```