

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **LAG and HASH API’s** |
| **Authors** | **DELL** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **1/23/2015** |
| **SAI-Version** | **V0.9.2** |

**Contents**

[List of Changes i](#_Toc416072895)

[1 Overview 1](#_Toc416072896)

[2 Specification 1](#_Toc416072897)

[2.1 LAG 1](#_Toc416072898)

[2.2 HASH 4](#_Toc416072899)

[2.2.1 New Definitions 4](#_Toc416072900)

[2.2.2 New attributes added to existing SWITCH attributes 5](#_Toc416072901)

[2.2.3 Definitions to be removed from existing header file 6](#_Toc416072902)

[3 Examples 6](#_Toc416072903)

[3.1 LAG example 6](#_Toc416072904)

[3.1.1 Add Ports to a LAG 7](#_Toc416072905)

[3.1.2 Remove Ports from a LAG 7](#_Toc416072906)

[3.1.3 Set port list in LAG 8](#_Toc416072907)

[3.1.4 Remove a LAG 8](#_Toc416072908)

[3.2 LAG Hash example 8](#_Toc416072909)

[3.2.1 LAG Hash Algorithm 8](#_Toc416072910)

[3.2.2 LAG Hash Seed 9](#_Toc416072911)

[3.2.3 LAG Hash Fields 9](#_Toc416072912)

[3.2.4 Only MAC Based Fields 9](#_Toc416072913)

[3.2.5 Only IP Based fields 9](#_Toc416072914)

[3.2.6 MAC and IP based fields 10](#_Toc416072915)

[3.2.7 L4 header based fields 10](#_Toc416072916)

[3.3 ECMP Hash example 10](#_Toc416072917)

[3.3.1 ECMP Hash Algorithm 10](#_Toc416072918)

[3.3.2 ECMP Hash Seed 10](#_Toc416072919)

[3.3.3 ECMP Hash Fields 11](#_Toc416072920)

[4 Appendix 11](#_Toc416072921)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.2 | Proposal for LAG, HASH – Version 1 |  | 01/23/15 |
| 0.9.2 | Version 2   * Incorporated feedback from Mellanox. Appendix section has details of the changes * Added details on ordering a port addition to a LAG |  | 02/16/15 |
| 0.92 | Version 3   * Addressed comments from Microsoft. * Used unified object concept * Added portlist as attribute for lag * Default values for HASH Field attributes * Added examples * Added a LAG attribute for setting the LAG members |  | 04/07/15 |

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

The document describes the API to create, remove and update Link Aggregation Group (LAG) object. The create call return a LAG id which can be later used to manage the port membership of the LAG. The remove call destroy the LAG object. The add\_ports\_to\_lag and remove\_ports\_from\_lag adds and removes SAI port to and from the LAG.

In this proposal, LAG is defined as a simple container for a group of the SAI ports. The LAG object only manages its port membership. All its member port attributes such as MTU, default VLAN are managed individually. The SAI applications need to be aware the LAG membership and is responsible to keep a consistent view across all its member port attributes, for example all member ports should have same MTU and same default VLAN.

SAI, for a given set of member ports, should maintain the same flow-to-port mapping to the extent possible when the member ports gets deleted and added back.

The document also defines hash attributes that can be used to configure LAG.

# Specification

## LAG

typedef enum \_sai\_api\_t {

SAI\_API\_LAG= 12, /\* sai\_lag\_api\_t \*

}

typedef enum \_sai\_lag\_attr\_t {

/\* SAI port list [sai\_object\_list\_t] (MAXDATORY\_ON\_CREATE|CREATE\_AND\_SET)

\* The LAG must have at least one SAI port member at the creation time \*/

SAI\_LAG\_ATTR\_PORT\_LIST,

} sai\_lag\_attr\_t;

/\*

\* Routine Description:

\* Create LAG.

\* Arguments:

\* [out] lag\_id - LAG 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\_lag \_fn)(**

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

**\_In\_ uint32\_t attr\_count,**

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

**);**

/\*

\* Routine Description:

\* Remove LAG.

\*

\* Arguments:

\* [in] lag\_id - lag id

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

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

**);**

/\*

\* Routine Description:

\* Adds Port to LAG. The port list will be appended to the end of current port list

\*

\* Arguments:

\* [in] lag\_id - LAG id

\* [in] port\_count - number of ports

\* [in] port\_list - pointer to membership structures

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

**typedef sai\_status\_t (\*sai\_add\_ports\_to\_lag\_fn)(**

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

**\_In\_ const sai\_object\_id\_t port\_list**

**);**

/\*

\* Routine Description:

\* Remove Port from LAG

\*

\* Arguments:

\* [in] lag\_id - LAG id

\* [in] port\_count - number of ports

\* [in] port\_list - pointer to membership structures

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

**typedef sai\_status\_t (\*sai\_remove\_ports\_from\_lag\_fn)(**

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

**\_In\_ const sai\_object\_id\_t port\_list**

**);**

/\*

\* Routine Description:

\* Set LAG interface attribute

\*

\* Arguments:

\* [in]lag\_id– lag id

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

**\_In\_ sai\_objectt\_id\_t lag\_id,**

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

**);**

/\*

\* Routine Description:

\* Get LAG interface attribute

\*

\* Arguments:

\* [in] lag\_id- lag\_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\_lag \_attribute\_fn)(**

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

**\_In\_ uint32\_t attr\_count,**

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

**);**

/\*

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

\*/

**typedef struct \_sai\_lag\_api\_t**

**{**

**sai\_create\_lag\_fn   create\_lag;**

**sai\_remove\_lag\_ fn remove\_lag;**

**sai\_set\_lag\_attribute\_fn set\_lag\_attribute;**

**sai\_get\_lag\_attribute\_fn get\_lag\_attribute;**

**sai\_add\_ports\_to\_lag\_fn add\_ports\_to\_lag;**

**sai\_remove\_ports\_from\_lag\_fn remove\_ports\_from\_lag;**

**} sai\_lag\_api\_t;**

## HASH

ECMP and LAG shall use the system wide hash logic for flow distribution over the links. Hash configuration is defined per switch for each application like ECMP and LAG. Hash calculation will use following attributes for flow distribution over LAG and ECMP

* Hash Seed
* Hash algorithm
* Packet hash fields

Since Hash by itself is not an object and does not have a handle, we cannot have set/get API’s for hash using the Hash Type(LAG, ECMP) as an handle. Hence we would add these to the switch attributes duplicating entries for LAG and ECMP

### New Definitions

**Changes to saitypes.h**

typedef uint32\_t  sai\_switch\_hash\_seed\_t; /\* hash seed value\*/

/\*

\* List of hash fields to set

\*/

typedef struct \_sai\_switch\_hash\_fields\_t

{

uint32\_t num\_fields ;

sai\_switch\_hash\_field\_types\_t \*field\_list ;

} sai\_switch\_hash\_fields\_t ;

typedef union {

…

sai\_switch\_hash\_fields\_t hash\_field\_list;

} sai\_attribute\_value\_t;

**Changes to saiswitch.h**

typedef enum \_sai\_switch\_hash\_field\_types\_t  
{  
    SAI\_HASH\_SRC\_IP,      
    SAI\_HASH\_DST\_IP,          
    SAI\_HASH\_VLAN\_ID,

    SAI\_HASH\_IP\_PROTOCOL,        
    SAI\_HASH\_ETHERTYPE,       
    SAI\_HASH\_L4\_SOURCE\_PORT,   
    SAI\_HASH\_L4\_DEST\_PORT,  
    SAI\_HASH\_SOURCE\_MAC ,      
    SAI\_HASH\_DEST\_MAC ,         
 SAI\_HASH\_DEST\_PORT ,      
    SAI\_HASH\_SOURCE\_PORT        
     
} sai\_switch\_hash\_field\_types\_t;

typedef enum \_sai\_switch\_hash\_algo\_t {

SAI\_HASH\_XOR ,

SAI\_HASH\_CRC,

SAI\_HASH\_RANDOM,

SAI\_HASH\_SYMMETRIC            
} sai\_switch\_hash\_algo\_t;

### New attributes added to existing SWITCH attributes

typedef enum \_sai\_switch\_attr\_t

{

…

/\* Hash algorithm for all LAG in the switch[sai\_switch\_hash\_algo\_t]

\* (default to SAI\_HASH\_CRC)

\*/

SAI\_SWITCH\_ ATTR\_LAG\_HASH\_ALGO,

/\* Hash seed for all LAG in the switch[sai\_switch\_hash\_seed\_t]\*/

SAI\_SWITCH\_ATTR\_LAG\_HASH\_SEED,

/\* Hash fields for all LAG in the switch[sai\_switch\_hash\_fields\_t]

\* (default all fields in sai\_switch\_hash\_field\_types\_t are enabled)

\*/

SAI\_SWITCH\_ATTR\_LAG\_HASH\_FIELDS,

/\* Hash algorithm for all ECMP in the switch[sai\_switch\_hash\_algo\_t]

\* (default to SAI\_HASH\_CRC)

\*/

SAI\_SWITCH\_ ATTR\_ECMP\_HASH\_ALGO,

/\* Hash seed for all ECMP in the switch[sai\_switch\_hash\_seed\_t]\*/

SAI\_SWITCH\_ATTR\_ECMP\_HASH\_SEED,

/\* Hash fields for all ECMP in the switch[sai\_switch\_hash\_fields\_t]

\* (default all fields in sai\_switch\_hash\_field\_types\_t are enabled)

\*/

SAI\_SWITCH\_ATTR\_ECMP\_HASH\_FIELDS,

….

} sai\_switch\_attr\_t;

### Definitions to be removed from existing header file

ECMP hashing type [sai\_switch\_ecmp\_hash\_type\_t] \*/

SAI\_SWITCH\_ATTR\_ECMP\_HASH\_TYPE,

/\* ECMP hashing fields [sai\_switch\_ecmp\_hash\_fields\_t] \*/

SAI\_SWITCH\_ATTR\_ECMP\_HASH\_FIELDS,

/\*

\* Attribute data for SAI\_SWITCH\_ECMP\_HASH\_TYPE

\*/

typedef enum \_sai\_switch\_ecmp\_hash\_type\_t

{

SAI\_SWITCH\_ECMP\_HASH\_TYPE\_XOR,

SAI\_SWITCH\_ECMP\_HASH\_TYPE\_CRC,

} sai\_switch\_ecmp\_hash\_type\_t;

typedef enum \_sai\_switch\_ecmp\_hash\_fields\_t

{

SAI\_SWITCH\_ECMP\_HASH\_SRC\_IP = (1 << 0),

SAI\_SWITCH\_ECMP\_HASH\_DST\_IP = (1 << 1),

SAI\_SWITCH\_ECMP\_HASH\_L4\_SRC\_PORT = (1 << 2),

SAI\_SWITCH\_ECMP\_HASH\_L4\_DST\_PORT = (1 << 3),

} sai\_switch\_ecmp\_hash\_fields\_t;

# Examples

## LAG example

**Create a LAG:**

To create a lag, a pointer of type sai\_object\_id\_t is passed. The lag Id will be out parameter which should be used by the application for further configurations on this LAG. The list of ports that are added to a LAG should be passed as argument. The list must contain at least one port.

For instance if a LAG is created with 4 ports

sai\_object\_id\_t lag\_id ;

sai\_port\_list\_t port\_list;

sai\_attribute\_t attr;

port\_list.num\_fields = 4;

port\_list.field\_list = malloc(sizeof(sai\_port\_list\_t)\*4);

port\_list.field\_list[0] = 1 ; //some valid port numbers

port\_list.field\_list[1] = 2 ;

port\_list.field\_list[2] = 3;

port\_list.field\_list[3] = 4 ;

attr.id = SAI\_LAG\_ATTR\_PORT\_LIST;

attr.value.portlist = port\_list;

create\_lag(&lag\_id, 1, (const attr\*)&attr) ;

The lag\_id would contain the ID for this specific lag that was created using the create call.

### Add Ports to a LAG

To add a list of ports to the LAG, a port list is created as shown below. It’s filled with the list of ports and passed to the add\_ports\_to\_lag as shown below. For example if ports 5,6 need to be added to LAG then the following programing is done.

sai\_port\_list\_t port\_list;

port\_list.num\_fields = 2;

port\_list.field\_list = malloc(sizeof(sai\_port\_list\_t)\*2);

port\_list.field\_list[0] = 5 ; //some valid port numbers

port\_list.field\_list[1] = 6 ;

add\_ports\_to\_lag(lag\_id, 2, (const object\_id)port\_list);

The add call is incremental. For example if ports 5,6 need to be added to the LAG, then for the next add\_ports\_to\_lag call only 5,6 ports can be passed in the port list instead of entire list of ports to the LAG.

### Remove Ports from a LAG

Below is the mechanis to remove a list of ports from a LAG. For example if ports 1,2 need to be removed from a LAG, then a port list if created and its passed to the remove\_ports\_from\_lag API along with the LAG ID.

sai\_port\_list\_t port\_list;

port\_list.num\_fields = 2;

port\_list.field\_list = malloc(sizeof(sai\_port\_list\_t)\*2);

port\_list.field\_list[0] = 1 ; //some valid port numbers

port\_list.field\_list[1] = 2 ;

remove\_ports\_from\_lag(lag\_id, 2, (const object\_id)port\_list);

### Set port list in LAG

The list of ports in the LAG can be set using set attribute SAI\_LAG\_ATTR\_PORT\_LIST. This will replace the existing ports in the LAG with the new list passed. For example if the lag list of ports should be replaced with 5,6,7,8 then below programming can be done.

sai\_object\_id\_t lag\_id ;

sai\_port\_list\_t port\_list;

sai\_attribute\_t attr;

port\_list.num\_fields = 4;

port\_list.field\_list = malloc(sizeof(sai\_port\_list\_t)\*4);

port\_list.field\_list[0] = 5 ; //some valid port numbers

port\_list.field\_list[1] = 6 ;

port\_list.field\_list[2] = 7;

port\_list.field\_list[3] = 8 ;

attr.id = SAI\_LAG\_ATTR\_PORT\_LIST;

attr.value.portlist = port\_list;

set\_lag\_attribute(lag\_id, (const sai\_attribute\_t\*)&attr);

Similarly using get attribute for SAI\_LAG\_ATTR\_PORT\_LIST the list of ports in a LAG can be obtained

### Remove a LAG

To remove a LAG completely

remove\_lag (lag\_id) ;

## LAG Hash example

sai\_attribute\_t attr;

### LAG Hash Algorithm

Example to set the LAG Hash algorithm to CRC:

attr.id = SAI\_SWITCH\_ ATTR\_LAG\_HASH\_ALGO;

attr.value. s32 = SAI\_HASH\_CRC;

set\_switch\_attribute((const attr\*)&attr);

Example to set the LAG Hash algorithm to XOR:

attr.id = SAI\_SWITCH\_ ATTR\_LAG\_HASH\_ALGO;

attr.value. s32 = SAI\_HASH\_XOR;

set\_switch\_attribute((const attr\*)&attr);

Similarly random and symmetric hash can be set by using SAI\_HASH\_RANDOM and SAI\_HASH\_SYMMETRIC respectively.

### LAG Hash Seed

Example to set a random number as Hash Seed:

attr.id = SAI\_SWITCH\_ ATTR\_LAG\_HASH\_SEED;

attr.value. u32 = 12321; //Some number for seed

set\_switch\_attribute((const attr\*)&attr);

### LAG Hash Fields

The LAG Hash fields denote the fields in the packet headers based on which the hash can be calculated and flows can be classified. The hash is then used to select a member port in the LAG using which the flow egresses. Various fields that can be set are available in sai\_switch\_hash\_field\_types\_t. Based on type of traffic that is expected to pass through the LAG the user can set the fields that might classify the flows and provide uniform distribution. The more the fields, higher the granularity.

### Only MAC Based Fields

Fill all the hash fields in the list to be set. For instance if the flows are based on MAC address then source MAC and destination MAC needs to be set in the list as shown below.

attr.id = SAI\_SWITCH\_ ATTR\_LAG\_HASH\_FIELDS

sai\_switch\_hash\_fields\_t hash\_field\_list;

hash\_field\_list.num\_fields = 2;

hash\_field\_list.field\_list = malloc(sizeof(sai\_switch\_hash\_field\_types\_t)\*2);

hash\_field\_list.field\_list[0] = SAI\_HASH\_SOURCE\_MAC ;

hash\_field\_list.field\_list[1] = SAI\_HASH\_DEST\_MAC ;

attr.value.hash\_field\_list = hash\_field\_list;

set\_switch\_attribute((const attr\*)&attr)

Apart from source MAC and destination MAC for layer2 based hashing SAI\_HASH\_VLAN\_ID, SAI\_HASH\_ETHERTYPE can be used to achieve more granularity

### Only IP Based fields

If the flows are based on IPV4 or IPV6 then the following field types need to be set SAI\_HASH\_SRC\_IP,

SAI\_HASH\_DST\_IP. To have more granularity SAI\_HASH\_IP\_PROTOCOL can be also set. Apart from it SAI\_HASH\_VLAN\_ID can be used if routing on VLAN is enabled and used.

### MAC and IP based fields

To achieve a finer granularity both MAC and IP based fields can be set in the above example SAI\_HASH\_SRC\_IP, SAI\_HASH\_DST\_IP, SAI\_HASH\_SOURCE\_MAC, SAI\_HASH\_DEST\_MAC  and SAI\_HASH\_DEST\_MAC  can be used. Additionally SAI\_HASH\_IP\_PROTOCOL and SAI\_HASH\_ETHERTYPE can be added if the flows need to be classified based on type of protocol.

### L4 header based fields

To achieve layer4 level granularity for the flows SAI\_HASH\_L4\_SOURCE\_PORT, SAI\_HASH\_L4\_DEST\_PORT can be used along with IP based fields. This can provide granularity at TCP/UDP header based level to classify the flows.

Apart from the above specified fields source port and destination port can be used which can provide additional granularity in classifying the flows SAI\_HASH\_DEST\_PORT and SAI\_HASH\_SOURCE\_PORT can be used with the above combinations.

During npu initialization the attribute SAI\_SWITCH\_ ATTR\_LAG\_HASH\_FIELDS as well as SAI\_SWITCH\_ ATTR\_ECMP\_HASH\_FIELDS would be set with the entire list of fields present in sai\_switch\_hash\_field\_types\_t.

If the hash fields are modified dynamically then the existing flows might get disrupted since hash would get recalculated and a new member different than the current one might be chosen for the flow.

## ECMP Hash example

sai\_attribute\_t attr;

### ECMP Hash Algorithm

To set ECMP has algorithm use the below attribute and assign the required hash algorithm from sai\_switch\_hash\_algo\_t

attr.id = SAI\_SWITCH\_ ATTR\_ECMP\_HASH\_ALGO;

attr.value. s32 = SAI\_HASH\_XOR;

set\_switch\_attribute((const attr\*)&attr);

### ECMP Hash Seed

Any random number can be assigned as seed for ECMP hash as shown below.

attr.id = SAI\_SWITCH\_ ATTR\_ECMP\_HASH\_SEED;

attr.value. u32 = 12321; //Some number for seed

set\_switch\_attribute((const attr\*)&attr);

### ECMP Hash Fields

ECMP hash fields can be set using the below attribute. The required fields can be filled in the list that is supplied to set the attribute. For example If source IP, Destination IP and VLAN ID are required to be set for hash, then the list is filled as below

attr.id = SAI\_SWITCH\_ ATTR\_ECMP\_HASH\_FIELDS

sai\_switch\_hash\_fields\_t hash\_field\_list;

hash\_field\_list.num\_fields = 3;

hash\_field\_list.field\_list = malloc(sizeof(sai\_switch\_hash\_field\_types\_t)\*3);

hash\_field\_list.field\_list[0] = SAI\_HASH\_SRC\_IP;

hash\_field\_list.field\_list[1] = SAI\_HASH\_DST\_IP;

hash\_field\_list.field\_list[0] = SAI\_HASH\_SRC\_IP;

hash\_field\_list.field\_list[1] = SAI\_HASH\_VLAN\_ID;

attr.value.hash\_field\_list = hash\_field\_list;

set\_switch\_attribute((const attr\*)&attr);

# Appendix

**Review Comments – Matty, Mellanox – 01/30/2015**

From: opencompute-networking-bounces@lists.opencompute.org [mailto:opencompute-networking-bounces@lists.opencompute.org] On Behalf Of Manickam, Arunsubash  
Sent: Saturday, January 31, 2015 5:43 AM  
To: mattyk@mellanox.com; opencompute-networking@lists.opencompute.org  
Subject: Re: [Opencompute-networking] SAI - LAG, HASH API Proposal

Dell - Internal Use - Confidential

Thanks for your feedback Matty

1 –

Remove\_all API -

I this out API should look the same since we don’t have delete\_all in router, net hop, neigh …

I suggest we remove it from LAG has well

Agreed I will remove it in my next update

2 –

All those HASH algo look like vendor specific implementation and not generic

I think we should it should be

SAI\_HASH\_CRC

SAI\_HASH\_XOR

SAI\_HASH\_RANDOM

In addition should have another HASH attribute SAI\_HASH\_ SYMMETRIC

In order to be able to use symmetric hash

Yes I agree we can reduce the hash list. Do you think the 3 above + Symmetric should be bare minimum we expect from a NPU. How about load based hashing

3 –

Some NPU support LAG hash per LAG

And ECMP hash per VRF /route

IMO we should have those attribute on LAG,VRF as well .

Yes we could add that in future. In that case the same set of attributes get duplicated or we define a attribute that is common. One suggestion would help in this case is to define HASH in a separate header file and have functions where HASH can be set for different objects – Switch, LAG, VRF. That would look clean and avoid duplicating the definitions

4 –

I think that we should have a single attribute for  hash fields

SAI\_SWITCH\_ ATTR\_LAG\_HASH\_FIELDS

SAI\_SWITCH\_ ATTR\_ECMP\_HASH\_FIELDS

In order to configure hash  on MAC,ipV4, MPLS,,Vxlan all you need to do it to call the set attribute with list on attribute

Agreed I will make the change

5 –

IMO we should not use  bit field

The way SAI should handle multiple value on thesame attribute

is by providing a list of  e attribute.(to same approch we took in ACL when use rule can match on multiple fields)

Yes we could follow the same approach as ACL to be consistent

Once we converge on #2 and #3 I will make the updates and send a revised proposal

Thanks,

Arun