

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **IP multicast** |
| **Authors** | **CENTEC** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **17/10/2015** |
| **SAI-Version** | **0.9.3** |

**Contents**

[List of Changes i](#_Toc440875976)

[1 Overview 1](#_Toc440875977)

[2 Specification 1](#_Toc440875978)

[2.1 Changes to sai.h 1](#_Toc440875979)

[2.2 Change to sainexthop.h 1](#_Toc440875980)

[2.3 Change to sainexthopgroup.h 1](#_Toc440875981)

[2.4 Change to saiswitch.h 2](#_Toc440875982)

[2.5 Change to saihostif.h 2](#_Toc440875983)

[2.6 New definitions in saiipmc.h 2](#_Toc440875984)

[3 Examples 5](#_Toc440875985)

[3.1 Create nexthop 6](#_Toc440875986)

[3.2 Create nexthop group 6](#_Toc440875987)

[3.3 Create IPMC entry 6](#_Toc440875988)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.3 | Base version |  | 17/10/2015 |

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

IP Multicast provides a solution to eliminate traffic redundancy and reduce server and CPU loads. This document covers details of IP multicast object –IP multicast group creating, IP multicast group removing, IP multicast group attributes setting and getting, RPF interface adding to IP multicast group, and RPF interface removing from IP multicast group.

# Specification

[This section describes the details of the proposed interface/API]

## Changes to sai.h

/\*

typedef enum \_sai\_api\_t {

SAI\_API\_IPMC= 45, /\* sai\_ipmc\_api\_t \*/

} sai\_api\_t;

## Change to sainexthop.h

/\*\*

\* @brief Next hop type

\*/

typedef enum \_sai\_next\_hop\_type\_t

{

SAI\_NEXT\_HOP\_IP,

/\*\* MPLS(NHLFE) next hop \*/

SAI\_NEXT\_HOP\_MPLS,

/\*\* tunnel next hop \*/

SAI\_NEXT\_HOP\_TUNNEL\_ENCAP,

/\*\* next hop for ipmc group member\*/

SAI\_NEXT\_HOP\_IPMC

} sai\_next\_hop\_type\_t;

## Change to sainexthopgroup.h

/\*

\* Next hop group type

\*/

typedef enum \_sai\_next\_hop\_group\_type\_t

{

/\*\* Next hop group is ECMP \*/

SAI\_NEXT\_HOP\_GROUP\_ECMP,

/\*\* Next hop group is IP multicast \*/

SAI\_NEXT\_HOP\_GROUP\_IPMC,

/\* Other types of next hop group to be defined in the future, e.g., WCMP \*/

} sai\_next\_hop\_group\_type\_t;

## New definitions in saiipmc.h

/\*\*

\* @brief Enum defining ipmc entry types.

\*/

typedef enum \_sai\_ipmc\_entry\_type\_t

{

/\*\*IPMC entry with type (S,G) \*/

SAI\_IPMC\_TYPE\_SG = 0x00000001,

/\*\* IPMC entry with type (\*,G)\*/

SAI\_IPMC\_TYPE\_XG = 0x00000002,

} sai\_ipmc\_entry\_type\_t;

/\*

\* IP multicast entry key

\*/

typedef struct \_sai\_ipmc\_entry\_t

{

sai\_object\_id\_t vrf\_id;

sai\_ip\_address\_t destination;

sai\_ip\_address\_t source;

} sai\_ipmc\_entry\_t;

/\*

\* Attribute Id for IP multicast entry

\*/

typedef enum \_sai\_ipmc\_entry\_attr\_t

{

/\* READ-WRITE \*/

/\*\* RPF router interface list [sai\_object\_id\_t] \*/

SAI\_IPMC\_ATTR\_RPF\_ROUTER\_INTERFACE\_LIST,

/\*\* Next hop group id for the packet [sai\_object\_id\_t]

\* The next hop id type should be SAI\_NEXT\_HOP\_GROUP\_IPMC

\* (MANDATORY\_ON\_CREATE|CREATE\_ONLY) \*/

SAI\_IPMC\_ATTR\_NEXT\_HOP\_GROUP\_ID,

/\* Custom range base value \*/

SAI\_IPMC\_ATTR\_CUSTOM\_RANGE\_BASE = 0x10000000

} sai\_ipmc\_entry\_attr\_t;

/\*

\* IPMC event type

\*/

typedef enum sai\_ipmc\_event\_t

{

/\*\* IPMC entry aged \*/

SAI\_IPMC\_EVENT\_AGED,

} sai\_ipmc\_event\_t;

/\*\* Notification data format received from SAI IPMC callback\*/

typedef struct \_sai\_ipmc\_event\_notification\_data\_t {

sai\_ipmc\_event\_t event\_type;

sai\_ipmc\_entry\_t ipmc\_entry;

} sai\_ipmc\_event\_notification\_data\_t;

/\*\*

\* Routine Description:

\* IPMC notifications

\*

\* Arguments:

\* [in] count - number of notifications

\* [in] data - pointer to ipmc event notification data array

\*

\* Return Values:

\* None

\*/

typedef void (\*sai\_ipmc\_event\_notification\_fn)(

\_In\_ uint32\_t count,

\_In\_ sai\_ipmc\_event\_notification\_data\_t \*data

);

/\*

\* Routine Description:

\* Create IP multicast entry

\*

\* Arguments:

\* [in] ipmc\_entry - IP multicast entry

\* [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\_ipmc\_entry\_fn)(

\_In\_ const sai\_ipmc\_entry\_t\* ipmc\_entry,

\_In\_ uint32\_t attr\_count,

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

);

/\*

\* Routine Description:

\* Remove IP multicast entry

\*

\* Arguments:

\* [in] ipmc\_entry - IP multicast entry

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_ipmc\_entry\_fn)(

\_In\_ const sai\_ipmc\_entry\_t\* ipmc\_entry

);

/\*

\* Routine Description:

\* Set IP multicast entry attribute value

\*

\* Arguments:

\* [in] IP multicast - IP multicast entry

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_ipmc\_entry\_attribute\_fn)(

\_In\_ const sai\_ipmc\_entry\_t\* ipmc\_entry,

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

);

/\*

\* Routine Description:

\* Get IP multicast entry attribute value

\*

\* Arguments:

\* [in] ipmc\_entry - IP multicast entry

\* [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\_ipmc\_entry\_attribute\_fn)(

\_In\_ const sai\_ipmc\_entry\_t\* ipmc\_entry,

\_In\_ uint32\_t attr\_count,

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

);

/\*

\* Routine Description:

\* add RPF interface list to IP multicast

\*

\* Arguments:

\* [in] ipmc\_entry - IP multicast entry

\* [in] router interface list - router interface list

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_add\_rpf\_interface\_to\_ipmc\_fn)(

\_In\_ const sai\_ipmc\_entry\_t\* ipmc\_entry,

\_In\_ const sai\_object\_list\_t \*router\_interface\_list

);

/\*

\* Routine Description:

\* remove RPF interface list from IP multicast

\*

\* Arguments:

\* [in] ipmc\_entry - IP multicast entry

\* [in] router interface list - router interface list

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_rpf\_interface\_from\_ipmc\_fn)(

\_In\_ const sai\_ipmc\_entry\_t\* ipmc\_entry,

\_In\_ const sai\_object\_list\_t \*router\_interface\_list

);

/\*

\* IP multicast method table retrieved with sai\_api\_query()

\*/

typedef struct \_sai\_ipmc\_api\_t

{

sai\_create\_ipmc\_entry\_fn create\_ipmc\_entry;

sai\_remove\_ipmc\_entry\_fn remove\_ipmc\_entry;

sai\_add\_rpf\_interface\_to\_ipmc\_fn add\_rpf\_interface\_to\_ipmc;

sai\_remove\_rpf\_interface\_from\_ipmc\_fn remove\_rpf\_interface\_from\_ipmc;

sai\_set\_ipmc\_entry\_attribute\_fn set\_ipmc\_entry\_attribute;

sai\_get\_ipmc\_entry\_attribute\_fn get\_ipmc\_entry\_attribute;

} sai\_ipmc\_api\_t;

## Change to saiswitch.h

/\*\*

\* @brief Switch notification table passed to the adapter via sai\_initialize\_switch()

\*/

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\_ipmc\_event\_notification\_fn on\_ipmc\_event;

} sai\_switch\_notification\_t;

## Change to saisrouter.h

/\*\*

\* Attribute Id in sai\_set\_virtual\_router\_attribute() and

\* sai\_get\_virtual\_router\_attribute() calls

\*/

typedef enum \_sai\_virtual\_router\_attr\_t

{

/\*\* READ-WRITE \*/

/\*\* IPMC entry aging time in seconds [sai\_uint32\_t]

\* Zero means aging is disabled.

\* (default to zero)

\*/

SAI\_VIRTUAL\_ROUTER\_ATTR\_IPMC\_AGING\_TIME,

} sai\_virtual\_router\_attr\_t;

## Change to saihostif.h

typedef enum \_sai\_hostif\_trap\_id\_t

{

/\*\* ipmc packets with RPF check fail

\* (default packet action is trap) \*/

SAI\_HOSTIF\_TRAP\_ID\_IPMC\_RPF\_FAIL = 0x00004002,

SAI\_HOSTIF\_TRAP\_ID\_CUSTOM\_EXCEPTION\_RANGE\_BASE = 0x00005000,

} sai\_hostif\_trap\_id\_t;

# Examples

Router interface should be created first, for example, rif1\_object, rif2\_object and rpf\_object have been created.

## Create nexthop for router rif1 and rif2

sai\_api\_query(SAI\_API\_NEXT\_HOP, & sai\_nexthop\_api);

sai\_object\_id\_t nh1\_object;

sai\_attribute\_t attr[2] = {0};

attr[0].id = SAI\_NEXT\_HOP\_ATTR\_TYPE;

attr[0].value.oid = SAI\_NEXT\_HOP\_IPMC;

attr[1].id = SAI\_NEXT\_HOP\_ATTR\_ROUTER\_INTERFACE\_ID;

attr[1].value.oid = rif1\_object;

sai\_nexthop\_api-> create\_next\_hop (&nh1\_object, 2, attr);

sai\_object\_id\_t nh2\_object;

sai\_attribute\_t attr[2] = {0};

attr[0].id = SAI\_NEXT\_HOP\_ATTR\_TYPE;

attr[0].value.oid = SAI\_NEXT\_HOP\_IPMC;

attr[1].id = SAI\_NEXT\_HOP\_ATTR\_ROUTER\_INTERFACE\_ID;

attr[1].value.oid = rif2\_object;

sai\_nexthop\_api->create\_next\_hop (&nh2\_object, 2, attr);

## Create nexthop group

sai\_api\_query(SAI\_API\_NEXT\_HOP\_GROUP, &sai\_nexthop\_group\_api);

sai\_object\_id\_t nexthop\_group\_object;

sai\_object\_id\_t nexthop\_list\_obj[2] = {0};

sai\_attribute\_t attr[2] = {0};

attr[0].id = SAI\_NEXT\_HOP\_GROUP\_ATTR\_TYPE;

attr[0].value.oid = SAI\_NEXT\_HOP\_GROUP\_IPMC;

attr[1].id = SAI\_NEXT\_HOP\_GROUP\_ATTR\_NEXT\_HOP\_LIST;

nexthop\_list\_obj [0] = nh1\_object;

nexthop\_list\_obj [1] = nh2\_object;

attr[1].value.objlist.count = 2;

attr[1].value.objlist.list = nexthop\_list\_obj;

sai\_nexthop\_group\_api->create\_next\_hop\_group(&nexthop\_group\_object, 2, attr);

## Create IPMC entry

sai\_api\_query(SAI\_API\_IPMC, &sai\_ipmc\_api);

sai\_ipmc\_entry\_t ipmc\_entry;

sai\_object\_id\_t rpf\_interface\_list\_obj[2] = {0};

sai\_attribute\_t attr[2] = {0};

ipmc\_entry.vrf\_id = vr\_id\_obj;

ipmc\_entry.destination.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

ipmc\_entry.destination.addr.ip4 = 0xE0010101;

ipmc\_entry.destination.mask.ip4 = 0xFFFFFFFF;

ipmc\_entry.destination.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

ipmc\_entry.source.addr.ip4 = 0XC0A80001;

ipmc\_entry.source.mask.ip4 = 0xFFFFFFFF;

attr[0].id = SAI\_IPMC\_ATTR\_NEXT\_HOP\_GROUP\_ID;

attr[0].value.oid = nexthop\_group\_object;

attr[1].id = SAI\_IPMC\_ATTR\_RPF\_ROUTER\_INTERFACE\_LIST;

rpf\_interface\_list\_obj[0] = rpf\_object;

attr[1].value.objlist.count = 1;

attr[1].value.objlist.list = rpf\_interface\_list\_obj;

sai\_ipmc\_api->create\_ipmc\_entry(&ipmc\_entry, 2, attr);

## IPMC event example

/\*example callback function, just send this information to protocol\*/

void example\_ipmc\_event\_notification(

uint32 count,

sai\_ipmc\_event\_notification\_data\_t \*noti\_data)

{

/\*send this notification data to protocol software\*/

}

/\*register the callback function\*/

sai\_switch\_notification. on\_ipmc\_event = example\_ipmc\_event\_notification;

# Appendix (review meeting questions)

1. Suggest the multicast group entry to be pointers to a group structure instead of port itself. That group structure could be shared by L2 and L3 multicast, containing the port bitmap for L2 and maybe VLAN info for L3.

Answer: About IPMC application with IGMP Snooping, the logical is similar to IPUC on VLAN interface. In this case, there are no neighbor table lookup and unicast FDB table lookup logical. But it will involve multicast FDB table lookup instead.

1. What would be a IPMC event as defined? Any example?

Answer: IPMC event will notify a IPMC aging event to software. I will add a example to proposal.

1. How to determine if a packet would go to multicast?

Answer: the condition is that the packet is from router interface and the packets is a legal IP multicast packet. in this case, the packet should go to multicast process.

1. Please clarify the type of IPMC nexthop.

Answer: First, we add a new type nexthop group called SAI\_NEXT\_HOP\_GROUP\_ECMP. And second we define a new type of nexthop called SAI\_NEXT\_HOP\_IPMC. One IPMC nexthop group is composed by multiple ipmc nexthop.

1. Why it is defined as prefix for the IPMC address? Should it be specific 32 bits?

Answer: the original definition is used for (\*,G). now prefix is changed to sai\_ip\_address\_t.and we add a new type to differentiate (S,G) and (\*,G).

1. Please detail the possible attributes of nexthop group

Answer: please refer to sai\_ipmc\_entry\_attr\_t.currently SAI\_IPMC\_ATTR\_NEXT\_HOP\_GROUP\_ID is MANDATORY\_ON\_CREATE attribute. And another attribute is rpf\_interface\_list.

1. Please describe the logical pipeline of L2 and L3 multicast. Better to separate these two to make it clearer.

Answer: yes, will add L2 and L3 pipeline into <<pipleline.vsd>> on git.