

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **Host interface** |
| **Authors** | **Mellanox** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **2/5/2015** |
| **SAI-Version** | **0.9.2** |

**Contents**

[List of Changes i](#_Toc410907891)

[1 Overview 1](#_Toc410907892)

[1.1 Classification & Registration 1](#_Toc410907893)

[1.1.1 Control traffic identification 1](#_Toc410907894)

[1.1.2 Classification 1](#_Toc410907895)

[1.1.3 Registration 2](#_Toc410907896)

[1.2 Packet send and receive 2](#_Toc410907897)

[1.2.1 Packet send and receive - net device 2](#_Toc410907898)

[2 Specification 3](#_Toc410907899)

[2.1 Host interface functionality 3](#_Toc410907900)

[2.1.1 Host interface trap group 3](#_Toc410907901)

[2.1.2 Host interface trap id 4](#_Toc410907902)

[2.1.3 Host interface packet send and receive 7](#_Toc410907903)

[2.1.4 Host interface summery 9](#_Toc410907904)

[2.2 New Call Back function for packer receive 9](#_Toc410907905)

[2.3 New port attribute 10](#_Toc410907906)

[2.4 New router interface attribute 10](#_Toc410907907)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.2 | Base version |  | 2/5/2015 |

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

In the normal operation of a network, the network device CPU is required to handle packets. Most notably, control-plane traffic (System-to-System protocol coordination traffic - e.g. LACP, IGMP ARP … ) and device management traffic must reach the CPU. In addition, The CPU has limited resources and therefore can obviously handle only a limited amount of data. As a result, it is possible that the processing of some of the traffic being sent to the CPU may be delayed or, in extreme cases, some of the traffic maybe even dropped.

A system which is properly configured ensures that high priority traffic will experience less (if at all) drops and will get to the CPU sooner than any other lower priority classified traffic. On the software side the application should ensure that high classified traffic will be prioritized in CPU processing time over other traffic.

In order to guarantee proper operation of the switch, the traffic types being sent to the CPU must be handled according to a predefined policy. Within this policy, the priorities, the bandwidth (allowed rate) and the burstiness of each traffic type should be defined.

Host interface SAI, composed from two parts

Classification & Registration

## Classification & Registration

SAI provided an ability to identify, register and assign a QoS for control traffic.

### Control traffic identification

SAI assign a unique identifier **trap\_id** the each control traffic

There are three types of trap **id :**

**Control protocol trap id :** trap id for a well-known d control protocol E.g. STP OSPF..

**Pipeline exception trap id :** trap id for a n exception in the switch-router forwarding pipeline known E.g. router TTL=1, RPF …

**User define trap id :** application has the ability extend and define additional control traffic trap id in order to support a new or proprietary control protocol or to define an additional exception in the switch pipeline this capability is achieved by providing the ability to define trap-id via the switch-router pipeline ACL ,router …

### Classification

Assign a QoS for control traffic since in most cases a few trap-id share the same QoS attribute and in order to reduce amount of configuration needed for system bring up.

A new container will be created, **Trap\_group**

Trap group attribute:

Priory, rate limiter, HW Traffic class

More than one trap id can be mapped into a single trap group



### Registration

Application can register to receive a trap\_id in addition it can control he action to apply on the trap-id options are

Ignore – forward the trap-id as a regular packet

Trap – terminate pipeline and send packet to CPU

Mirror- sent a copy to the CPU the original packet will continue the pipeline

Discard – drop

## Packet send and receive

SAI provide three different channel in order to send and receive packet from and to the CPU

* OS network device
* a generic file socket
* function callback

### Packet send and receive - net device

SAI will provide the ability to receive (and send) trap-id on the operation system network device infra-structure .

And this is in order to enable standard application such as Quagga to operate the switch as if it is a host

ASI will provide the ability to create those net devises whether they represent a physical port or a L3 router interface

# Specification

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

## Host interface functionality

### Host interface trap group

typedef enum \_sai\_hostif\_group\_attr\_t

{

/\* Admin Mode [bool] \*/

SAI\_HOSTIF\_GROUP\_ATTR\_ADMIN\_STATE,

/\* group priorety [uint32\_t] \*/

SAI\_HOSTIF\_GROUP\_ATTR\_PRIO,

/\* group egress traffic class [uint32\_t] \*/

SAI\_HOSTIF\_GROUP\_ATTR\_TC,

/\* group policer id [uint32\_t] \*/

SAI\_HOSTIF\_GROUP\_ATTR\_POLICER,

SAI\_HOSTIF\_GROUP\_CUSTOM\_RANGE\_BASE = 0x10000000

} sai\_hostif\_group\_attr\_t;

/\*

\* Routine Description:

\* create host interface group .

\*

\* Arguments:

\* [in,out] hostif\_group\_id - host intarfec group 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\_hostif\_group\_fn)(

\_In\_out sai\_hostif\_group\_id\_t\* hostif\_group\_id,

\_In\_ int attr\_count,

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

);

/\*

\* Routine Description:

\* delete host interface group .

\*

\* Arguments:

\* [in] hostif\_group\_id - host intarfec group id

\*

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_hostif\_group\_fn)(

\_in\_ sai\_hostif\_group\_id\_t hostif\_group\_id,

);

/\*

\* Routine Description:

\* Set host interface group attribute value.

\*

\* Arguments:

\* [in] hostif\_group\_id - host intarfec group 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\_set\_hostif\_group\_attribute\_fn )

(

\_in\_ sai\_hostif\_group\_id\_t hostif\_group\_id,

\_In\_ int attr\_count,

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

);

/\*

\* Routine Description:

\* get host interface group attribute value.

\*

\* Arguments:

\* [in] hostif\_group\_id - host intarfec group id

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

\* [in,out] attr\_list - array of attributes

\*

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_hostif\_group\_attribute\_fn )(

\_in\_ sai\_hostif\_group\_id\_t hostif\_group\_id,

\_In\_ int attr\_count,

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

);

### Host interface trap id

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

{

//control protocol

SAI\_HOSTIF\_TRAP\_ID\_STP,

SAI\_HOSTIF\_TRAP\_ID\_LACP,

SAI\_HOSTIF\_TRAP\_ID\_EAPOL,

SAI\_HOSTIF\_TRAP\_ID\_LLDP,

SAI\_HOSTIF\_TRAP\_ID\_PVRST,

SAI\_HOSTIF\_TRAP\_ID\_IGMP\_TYPE\_QUERY,

SAI\_HOSTIF\_TRAP\_ID\_IGMP\_TYPE\_V1\_REPORT,

SAI\_HOSTIF\_TRAP\_ID\_IGMP\_TYPE\_V2\_REPORT,

SAI\_HOSTIF\_TRAP\_ID\_IGMP\_TYPE\_V3\_REPORT,

SAI\_HOSTIF\_TRAP\_ID\_DHCP,

SAI\_HOSTIF\_TRAP\_ID\_ARP\_REQUEST,

SAI\_HOSTIF\_TRAP\_ID\_ARP\_RESPONSE,

SAI\_HOSTIF\_TRAP\_ID\_OSPF,

SAI\_HOSTIF\_TRAP\_ID\_PIM,

SAI\_HOSTIF\_TRAP\_ID\_VRRP,

SAI\_HOSTIF\_TRAP\_ID\_BGP,

SAI\_HOSTIF\_TRAP\_ID\_IPV6\_NEIGHBOR\_DISCOVERY,

SAI\_HOSTIF\_TRAP\_ID\_IPV6\_MLD\_V1\_V2 ,

SAI\_HOSTIF\_TRAP\_ID\_IPV6\_MLD\_V1\_REPORT,

SAI\_HOSTIF\_TRAP\_ID\_IPV6\_MLD\_V1\_DONE,

SAI\_HOSTIF\_TRAP\_ID\_MLD\_V2\_REPORT ,

//pipelie exception

SAI\_HOSTIF\_TRAP\_ID\_L3\_MTU\_ERROR,

SAI\_HOSTIF\_TRAP\_ID\_TTL\_ERROR,

  SAI\_HOSTIF\_TRAP\_ID\_L3\_RPF,

SAI\_HOSTIF\_TRAP\_ID\_L3\_ASSERT,

//user define trap

SAI\_HOSTIF\_TRAP\_ID\_ROUTER,

SAI\_HOSTIF\_TRAP\_ID\_NEIGH,

SAI\_HOSTIF\_TRAP\_ID\_FDB,

SAI\_HOSTIF\_TRAP\_ID\_ACL,

} sai\_hostif\_trap\_id\_t;

typedef enum \_sai\_hostif\_action\_t

{

SAI\_HOSTIF\_FORWARD,

SAI\_HOSTIF\_*TRAP*,

SAI\_HOSTIF\_ *MIRROR*,

SAI\_HOSTIF\_ *DISCARD*,

SAI\_HOSTIF\_CUSTOM\_RANGE\_BASE = 0x10000000

}sai\_hostif\_action\_t

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

{

/\* Admin Mode enable/diasble trap [bool] \*/

SAI\_HOSTIF\_TRAP\_ID\_ADMIN\_STATE,

/\* trap channel to use [sai\_hostif\_trap\_channel\_t] \*/

SAI\_HOSTIF\_TRAPID\_ATTR\_TRAP\_CHANNEL,

/\* valid only when SAI\_HOSTIF\_TRAPID\_ATTR\_TRAP\_CHANNEL=SAI\_HOSTIF\_CHANNEL\_TYPE\_FD\*/

SAI\_HOSTIF\_TRAPID\_ATTR\_FD,

/\* optional enable trap in a specific port (default is global)\*/

SAI\_HOSTIF\_TRAPID\_ATTR\_PORT,

/\* trap action [sai\_hostif\_action\_t] \*/

SAI\_HOSTIF\_TRAPID\_ATTR\_TRAP\_ACTION,

SAI\_HOSTIF\_TRAPID\_CUSTOM\_RANGE\_BASE = 0x10000000

} sai\_hostif\_trap\_id\_attr\_t

/\*

\* Routine Description:

\* Set trap id attribute value.

\*

\* Arguments:

\* [in] hostif\_trap\_id - host intarfec trap 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\_set\_hostif\_trapid\_attribute\_fn )(

\_in\_ sai\_hostif\_trap\_id\_t hostif\_trapid,

\_In\_ int attr\_count,

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

);

/\*

\* Routine Description:

\* Get trap id attribute value.

\*

\* Arguments:

\* [in] hostif\_trap\_id - host intarfec trap id

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

\* [in,out] attr\_list - array of attributes

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_hostif\_trapid\_attribute\_fn )(

\_in\_ sai\_hostif\_trap\_id\_t hostif\_trapid,

\_In\_ int attr\_count,

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

);

### Host interface packet send and receive

typedef enum \_sai\_tx\_type

{

SAI\_TX\_TYPE\_ PIPELINE\_LOOKUP,

SAI\_TX\_TYPE\_PIPELINE\_BYPASS,

SAI\_ TX\_TYPE\_CUSTOM\_RANGE\_BASE = 0x10000000

} sai\_tx\_type\_t

typedef enum \_sai\_packet\_attr

{

//ingress attributes

SAI\_PACKET\_TRAP\_ID,

SAI\_PACKET\_INGRESS\_PORT,

SAI\_PACKET\_INGRESS\_LAG,

SAI\_PACKET\_IS\_LAG,

//egrsss attributes

//Sent type [sai\_tx\_type\_t]

SAI\_PACKET\_TX\_TYPE,

SAI\_PACKET\_EGRESS\_PORT,

SAI\_PACKET\_EGRESS\_TC,

} sai\_packet\_attr\_t;

/\*

\* Routine Description:

\* create host interface channel .

\*

\* Arguments:

\* [in,out] hostif\_fd - host intarface file descriptor .

\*

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_create\_hostif\_channel\_fn)(

\_in\_out sai\_hostif\_fd\_t\* hostif\_fd,

);

/\*

\* Routine Description:

\* delete host interface channel .

\*

\* Arguments:

\* [in] hostif\_fd - host intarface file descriptor .

\*

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_remove\_hostif\_channel\_fn)(

\_in\_ sai\_hostif\_fd\_t hostif\_fd);

/\*

\* Routine Description:

\* hostif receive funtion only .

\*

\* Arguments:

\* [in] hostif\_fd - host intarface file descriptor

\* [out] buffer – packet buffer

\* [out] buffer size- packet size in bytes .

\* [out] attr\_count - number of attributes

\* [out] attr\_list - array of attributes

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\* sai\_recv\_hostif\_packet\_fn)(

\_in\_ sai\_hostif\_fd\_t hostif\_fd ,

\_out\_ void \* buffer ,

\_out\_ sai\_size\_t \* buffer\_size ,

\_out\_ int attr\_count,

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

);

/\*

\* Routine Description:

\* hostif send funtion .

\*

\* Arguments:

\* [in] hostif\_fd - host intarface file descriptor .

\* [In] buffer – packet buffer

\* [in] buffer size- packet size in bytes .

\* [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\_send\_hostif\_packet\_fn)(

\_in\_ sai\_hostif\_fd\_t hostif\_fd ,

\_in\_out void \* buffer ,

\_in\_out sai\_size\_t \* buffer\_size ,

\_In\_ int attr\_count,

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

);

### Host interface summery

/\*

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

\*/

typedef struct \_sai\_hostif\_api\_t

{

sai\_create\_hostif\_channel\_fn create\_hostif\_channel;

sai\_remove\_hostif\_channel\_fn remove\_hostif\_channel;

sai\_create\_hostif\_group\_fn create\_hostif\_group;

sai\_remove\_hostif\_group\_fn remove\_hostif\_group;

sai\_set\_hostif\_group\_attribute\_fn set\_group\_attribute;

sai\_get\_hostif\_group\_attribute\_fn get\_ group\_attribute;

sai\_set\_hostif\_trapid\_attribute\_fn set\_trapid\_attribute;

sai\_get\_hostif\_trapid\_attribute\_fn get\_trapid\_attribute;

sai\_recv\_hostif\_packet\_fn recv\_packet;

sai\_send\_hostif\_packet\_fn send\_packet;

} sai\_hostif\_api\_t;

## New Call Back function for packer receive

|  |
| --- |
| \* 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\_packet\_event\_notification\_fn on\_packet\_event;** |
| sai\_port\_state\_change\_notification\_fn on\_port\_state\_change; |
| sai\_switch\_shutdown\_request\_fn on\_switch\_shutdown\_request; |
| } sai\_switch\_notification\_t;  typedef void (\*sai\_packet\_event\_notification\_fn)(  void \* buffer ,  sai\_size\_t \* buffer\_size ,  int attr\_count,  sai\_attr\_t \* attr\_list  ); |
|  |

## New port attribute

typedef enum \_sai\_port\_attr\_t

{

/\* Port Type [sai\_port\_type\_t] \*/

SAI\_PORT\_ATTR\_TYPE,

|  |
| --- |
| /\* Operational Status [sai\_port\_oper\_status\_t] \*/ |
| SAI\_PORT\_ATTR\_OPER\_STATUS, |

…

/\* create net-device [bool] \*/

**SAI\_PORT\_ATTR\_CREATE\_NETDEVICE**

/\* net device name [char \*] \*/

**SAI\_PORT\_ATTR\_NETDEVICE\_NAME**

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

} sai\_port\_attr\_t;

## New router interface attribute

typedef enum sai\_router\_interface\_attr\_t

{

/\* Virtual router id [sai\_virtual\_router\_id\_t] \*/

SAI\_ROUTE\_VIRTUAL\_ROUTER\_ID,

/\* interface Type [sai\_router\_interface\_type\_t] \*/

SAI\_ROUTER\_INTERFACE\_ATTR\_TYPE

…

**/\* create router interface net-device [bool] \*/**

**SAI\_ROUTER\_INTERFACE\_ATTR\_CREATE\_NETDEVICE**

**/\* net device name [char \*] \*/**

**SAI\_ROUTER\_INTERFACE\_ATTR\_NETDEVICE\_NAME**

SAI\_ROUTER\_INTERFACE\_CUSTOM\_RANGE\_BASE = 0x10000000

} ;