

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **Host interface** |
| **Authors** | **Mellanox** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **3/9/2015** |
| **SAI-Version** | **0.9.2** |

**Contents**

[List of Changes i](#_Toc416314959)

[1 Overview 1](#_Toc416314960)

[1.1 Classification & Registration 1](#_Toc416314961)

[1.1.1 Control traffic identification 1](#_Toc416314962)

[1.1.2 Classification 1](#_Toc416314963)

[1.1.3 Registration 2](#_Toc416314964)

[1.2 Packet send and receive 2](#_Toc416314965)

[1.2.1 Packet send and receive - net device 2](#_Toc416314966)

[2 Specification 3](#_Toc416314967)

[2.1 Host interface functionality 3](#_Toc416314968)

[2.1.1 Host interface trap group 3](#_Toc416314969)

[2.1.2 Host interface trap 4](#_Toc416314970)

[2.1.3 Host interface user defined trap 8](#_Toc416314971)

[2.1.4 Host interface channel 10](#_Toc416314972)

[2.1.5 Host interface packet send and receive 12](#_Toc416314973)

[2.1.6 Host interface API 13](#_Toc416314974)

[2.2 New Callback function for packet receive 13](#_Toc416314975)

[2.3 Changes in saitypes.h 14](#_Toc416314976)

[2.4 Changes in saiswitch.h 14](#_Toc416314977)

[2.5 Configuration example – using netdev channel 14](#_Toc416314978)

[2.5.1 Step 1- create three trap groups 14](#_Toc416314979)

[2.5.2 Step 2- create host interface channel 15](#_Toc416314980)

[2.5.3 Step 3- configure the trap-ids 16](#_Toc416314981)

[2.6 Configuration example – using file descriptor channel 17](#_Toc416314982)

[2.6.1 Step 1- create three trap groups 17](#_Toc416314983)

[2.6.2 Step 2- create host interface FD 17](#_Toc416314984)

[2.6.3 Step 3- configure the trap-ids 18](#_Toc416314985)

[2.6.4 Step 4- send and receive 19](#_Toc416314986)

[2.7 Configuration example – using packet receive callback channel 19](#_Toc416314987)

[2.7.1 Step 1- create three trap groups 20](#_Toc416314988)

[2.7.2 Step 2- configure the trap-ids 20](#_Toc416314989)

[2.7.3 Step 4- send and receive 21](#_Toc416314990)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.9.2 | Base version |  | 2/5/2015 |
| 0.9.2 | Version 2 with review comments |  | 3/18/2015 |
| 0.9.2 | Version 3 with review comments |  | 4/7/2015 |
| 0.9.2 | Version 4 with review comments |  | 4/9/2015 |

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 may even be 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 provides an ability to identify, register and assign a QoS for control traffic.

### Control traffic identification

SAI assigns a unique identifier **trap\_id** to each control traffic

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

**Control protocol trap id:** trap id for well-known control protocol, e.g. STP OSPF …

**Pipeline exception trap id:** trap id for an exception in the switch-router forwarding pipeline, e.g. router TTL=1, RPF …

**User define trap id:** application has the ability to 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, e.g. 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 attributes:

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 the 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 provides three different channels in order to send and receive packet from and to the CPU

* OS network device
* a generic file socket
* send function and receive function / receive 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 infrastructure.

This is done in order to enable standard application such as Quagga to operate the switch as if it is a host.

SAI will provide the ability to create these net devices whether they represent a physical port or a L3 router interface.

On RX, the driver demux the packet to the relevant net device according to incoming port / RIF ID.

On TX, the application chooses which net device to use.

# 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] (default to TRUE) \*/

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

/\* group priority [uint32\_t] (MANDATORY\_ON\_CREATE). This is equivalent

\* to ACL table priority SAI\_ACL\_TABLE\_ATTR\_PRIORITY \*/

SAI\_HOSTIF\_GROUP\_ATTR\_PRIO,

/\* group egress queue [uint32\_t] (MANDATORY\_ON\_CREATE) \*/

SAI\_HOSTIF\_GROUP\_ATTR\_QUEUE,

/\* sai\_object\_id\_t [uint32\_t] \*/

SAI\_HOSTIF\_GROUP\_ATTR\_POLICER,

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

} sai\_hostif\_group\_attr\_t;

/\*

\* Routine Description:

\* Create host interface group

\*

\* Arguments:

\* [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)(

\_Out\_ sai\_object\_id\_t \*hostif\_group\_id,

\_In\_ uint32\_t attr\_count,

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

);

/\*

\* Routine Description:

\* Remove 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\_object\_id\_t hostif\_group\_id

);

/\*

\* Routine Description:

\* Set host interface group attribute value.

\*

\* Arguments:

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

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

(

\_In\_ sai\_object\_id\_t hostif\_group\_id,

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

);

/\*

\* 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\_object\_id\_t hostif\_group\_id,

\_In\_ uint32\_t attr\_count,

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

);

### Host interface trap

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

{

/\*

\* cdefault trap-id

\*/

SAI\_HOSTIF\_TRAP\_ID\_DEFAULT,

/\*

\* control protocol

\*/

/\*

\* switch trap

\*/

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\_LEAVE,

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\_SWITCH\_CUSTOM\_RANGE\_BASE = 0x10000000,

/\*

\* router trap

\*/

SAI\_HOSTIF\_TRAP\_ID\_ARP\_REQUEST = 0x20000000,

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

SAI\_HOSTIF\_TRAP\_ID\_DHCP,

SAI\_HOSTIF\_TRAP\_ID\_OSPF,

SAI\_HOSTIF\_TRAP\_ID\_PIM,

SAI\_HOSTIF\_TRAP\_ID\_VRRP,

SAI\_HOSTIF\_TRAP\_ID\_BGP,

SAI\_HOSTIF\_TRAP\_ID\_DHCPV6,

SAI\_HOSTIF\_TRAP\_ID\_OSPFV6,

SAI\_HOSTIF\_TRAP\_ID\_VRRPV6,

SAI\_HOSTIF\_TRAP\_ID\_BGPV6,

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 ,

SAI\_HOSTIF\_TRAP\_ID\_ROUTER\_CUSTOM\_RANGE\_BASE = 0x30000000,

/\*

\* pipeline exceptions

\*/

SAI\_HOSTIF\_TRAP\_ID\_L3\_MTU\_ERROR = 0x40000000,

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

  SAI\_HOSTIF\_TRAP\_ID\_L3\_RPF,

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

SAI\_HOSTIF\_TRAP\_ID\_CUSTOM\_EXCEPTION\_RANGE\_BASE = 0x50000000,

} sai\_hostif\_trap\_id\_t;

typedef enum \_sai\_hostif\_trap\_channel\_t

{

/\* receive packets via file desriptor \*/

SAI\_HOSTIF\_TRAP\_CHANNEL\_FD,

/\* receive packets via callback \*/

SAI\_HOSTIF\_TRAP\_CHANNEL\_CB,

/\* receive packets via OS net device \*/

SAI\_HOSTIF\_TRAP\_CHANNEL\_NETDEV,

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

} sai\_hostif\_trap\_channel\_t;

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

{

/\* Admin Mode enable/disable trap [bool] (default to TRUE) \*/

SAI\_HOSTIF\_TRAP\_ATTR\_ADMIN\_STATE,

/\* trap channel to use [sai\_hostif\_trap\_channel\_t] (MANDATORY\_ON\_CREATE) \*/

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

/\* file descriptor [sai\_object\_id\_t]

\* Valid only when SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL == SAI\_HOSTIF\_TRAP\_CHANNEL\_FD

\* (MANDATORY\_ON\_CREATE when SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL ==

\* SAI\_HOSTIF\_TRAP\_CHANNEL\_FD)

\*/

SAI\_HOSTIF\_TRAP\_ATTR\_FD,

/\* enable trap for specific ports (default is global) [sai\_port\_list\_t] \*/

SAI\_HOSTIF\_TRAP\_ATTR\_PORT\_LIST,

/\* trap action [sai\_packet\_action\_t] (MANDATORY\_ON\_CREATE) \*/

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

/\* trap-group ID for the trap [sai\_object\_id\_t] (MANDATORY\_ON\_CREATE) \*/ SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP,

/\* trap priority [uint32\_t] (default to 0). This is equivalent to

\* ACL entry priority SAI\_ACL\_ENTRY\_ATTR\_PRIORITY \*/

SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_PRIORITY,

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

} sai\_hostif\_trap\_attr\_t;

/\*

\* Routine Description:

\* Create host interface trap

\*

\* Arguments:

\* [in] hostif\_trap\_id - host interface 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\_create\_hostif\_trap\_fn)(

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

\_In\_ uint32\_t attr\_count,

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

);

/\*

\* Routine Description:

\* Remove host interface trap

\*

\* Arguments:

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

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

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

\_In\_ sai\_hostif\_trap\_id\_t hostif\_trapid

);

/\*

\* Routine Description:

\* Set trap attribute value.

\*

\* Arguments:

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

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_hostif\_trap\_attribute\_fn )(

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

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

);

/\*

\* Routine Description:

\* Get trap attribute value.

\*

\* Arguments:

\* [in] hostif\_trap\_id - host interface 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\_trap\_attribute\_fn )(

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

\_In\_ uint32\_t attr\_count,

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

);

### Host interface user defined trap

#define SAI\_HOSTIF\_USER\_DEFINED\_TRAP\_ID\_TABLE\_RANGE 0x1000

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

{

/\*

\* user defined traps default

\*/

SAI\_HOSTIF\_USER\_DEFINED\_TRAP\_ID\_DEFAULT,

/\*

\* user defined traps

\*/

SAI\_HOSTIF\_TRAP\_ID\_ROUTER\_MIN,

SAI\_HOSTIF\_TRAP\_ID\_ROUTER\_MAX=SAI\_HOSTIF\_TRAP\_ID\_ROUTER\_MIN + SAI\_HOSTIF\_USER\_DEFINED\_TRAP\_ID\_TABLE\_RANGE,

SAI\_HOSTIF\_TRAP\_ID\_NEIGH\_MIN,

SAI\_HOSTIF\_TRAP\_ID\_NEIGH\_MAX=SAI\_HOSTIF\_TRAP\_ID\_NEIGH\_MIN + SAI\_HOSTIF\_USER\_DEFINED\_TRAP\_ID\_TABLE\_RANGE,

SAI\_HOSTIF\_TRAP\_ID\_FDB\_MIN,

SAI\_HOSTIF\_TRAP\_ID\_FDB\_MAX=SAI\_HOSTIF\_TRAP\_ID\_FDB\_MIN + SAI\_HOSTIF\_USER\_DEFINED\_TRAP\_ID\_TABLE\_RANGE,

SAI\_HOSTIF\_TRAP\_ID\_SFLOW,

SAI\_HOSTIF\_TRAP\_ID\_CUSTOM\_RANGE\_BASE = 0x10000000,

} sai\_hostif\_user\_defined\_trap\_id\_t;

typedef enum \_sai\_hostif\_user\_defined\_trap\_attr\_t

{

/\* Admin Mode enable/disable trap [bool] (default to TRUE) \*/

SAI\_HOSTIF\_USER\_DEFINED\_TRAP\_ATTR\_ADMIN\_STATE,

/\* trap channel to use [sai\_hostif\_trap\_channel\_t] (MANDATORY\_ON\_CREATE) \*/

SAI\_HOSTIF\_USER\_DEFINED\_TRAP\_ATTR\_TRAP\_CHANNEL,

/\* file descriptor [sai\_object\_id\_t]

\* Valid only when SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL == SAI\_HOSTIF\_TRAP\_CHANNEL\_FD

\* (MANDATORY\_ON\_CREATE when SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL ==

\* SAI\_HOSTIF\_TRAP\_CHANNEL\_FD)

\*/

SAI\_HOSTIF\_USER\_DEFINED\_TRAP\_ATTR\_FD

} sai\_hostif\_user\_defined\_trap\_attr\_t;

/\*

\* Routine Description:

\* Create host interface user defined trap

\*

\* Arguments:

\* [in] hostif\_user\_defined\_trap\_id - host interface user defined 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\_create\_hostif\_user\_defined\_trap\_fn)(

\_In\_ sai\_hostif\_user\_defined\_trap\_id\_t hostif\_user\_defined\_trapid,

\_In\_ uint32\_t attr\_count,

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

);

/\*

\* Routine Description:

\* Remove host interface user defined trap

\*

\* Arguments:

\* [in] hostif\_user\_defined\_trap\_id - host interface user defined trap id

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_remove\_hostif\_user\_defined\_trap\_fn)(

\_In\_ sai\_hostif\_user\_defined\_trap\_id\_t hostif\_user\_defined\_trapid

);

/\*

\* Routine Description:

\* Set user defined trap attribute value.

\*

\* Arguments:

\* [in] hostif\_user\_defined\_trap\_id - host interface user defined trap id

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_hostif\_user\_defined\_trap\_attribute\_fn )(

\_In\_ sai\_hostif\_user\_defined\_trap\_id\_t hostif\_user\_defined\_trapid,

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

);

/\*

\* Routine Description:

\* Get user defined trap attribute value.

\*

\* Arguments:

\* [in] hostif\_user\_defined\_trap\_id - host interface user defined 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\_user\_defined\_trap\_attribute\_fn )(

\_In\_ sai\_hostif\_user\_defined\_trap\_id\_t hostif\_user\_defined\_trapid,

\_In\_ uint32\_t attr\_count,

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

);

### Host interface channel

/\*

\* Attribute data for SAI\_HOST\_INTERFACE\_ATTR\_TYPE

\*/

typedef enum \_sai\_host\_interface\_type\_t

{

/\* Port-based Host Interface Type \*/

SAI\_HOST\_INTERFACE\_TYPE\_PORT,

/\* Router Interface based Host Interface Type \*/

SAI\_HOST\_INTERFACE\_TYPE\_RIF,

/\* file descriptor \*/

SAI\_HOST\_INTERFACE\_TYPE\_FD

} sai\_host\_interface\_type\_t;

/\*

\* Host interface attribute IDs

\*/

typedef enum \_sai\_host\_interface\_attr\_t

{

/\* READ-ONLY \*/

/\* READ-WRITE \*/

/\* Type [sai\_host\_interface\_type\_t] (MANDATORY\_ON\_CREATE|CREATE\_ONLY) \*/

SAI\_HOST\_INTERFACE\_ATTR\_TYPE,

/\* Assosiated port or router interface [sai\_object\_id\_t]

\* Valid only when SAI\_HOST\_INTERFACE\_ATTR\_TYPE == SAI\_HOST\_INTERFACE\_TYPE\_PORT ||

\* SAI\_HOST\_INTERFACE\_TYPE\_RIF

\* (MANDATORY\_ON\_CREATE when SAI\_HOST\_INTERFACE\_ATTR\_TYPE ==

\* SAI\_HOST\_INTERFACE\_TYPE\_PORT || SAI\_HOST\_INTERFACE\_TYPE\_RIF | CREATE\_ONLY) \*/

SAI\_HOST\_INTERFACE\_ATTR\_PORT\_RIF\_ID,

/\* Name [char[HOST\_INTERFACE\_NAME\_SIZE]]

\* The maximum number of charactars for the name is HOST\_INTERFACE\_NAME\_SIZE - 1 since

\* it needs the terminating null byte ('\0') at the end.

\* Valid only when SAI\_HOST\_INTERFACE\_ATTR\_TYPE == SAI\_HOST\_INTERFACE\_TYPE\_PORT ||

\* SAI\_HOST\_INTERFACE\_TYPE\_RIF

\* (MANDATORY\_ON\_CREATE when SAI\_HOST\_INTERFACE\_ATTR\_TYPE ==

\* SAI\_HOST\_INTERFACE\_TYPE\_PORT || SAI\_HOST\_INTERFACE\_TYPE\_RIF) \*/

SAI\_HOST\_INTERFACE\_ATTR\_NAME,

/\* Custom range base value \*/

SAI\_HOST\_INTERFACE\_ATTR\_CUSTOM\_RANGE\_BASE = 0x10000000

} sai\_host\_interface\_attr\_t;

/\*

\* Routine Description:

\* Create host interface

\*

\* Arguments:

\* [out] hif\_id - host interface 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\_host\_interface\_fn)(

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

\_In\_ uint32\_t attr\_count,

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

);

/\*

\* Routine Description:

\* Remove host interface

\*

\* Arguments:

\* [in] hif\_id - host interface id

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t(\*sai\_remove\_host\_interface\_fn)(

\_In\_ sai\_object\_id\_t hif\_id

);

/\*

\* Routine Description:

\* Set host interface attribute

\*

\* Arguments:

\* [in] hif\_id - host interface id

\* [in] attr - attribute

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* Failure status code on error

\*/

typedef sai\_status\_t (\*sai\_set\_host\_interface\_attribute\_fn)(

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

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

);

/\*

\* Routine Description:

\* Get host interface attribute

\*

\* Arguments:

\* [in] hif\_id - host interface 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\_host\_interface\_attribute\_fn)(

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

\_In\_ uint32\_t attr\_count,

\_Inout\_ sai\_attribute\_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

{

/\* Trap ID [sai\_hostif\_trap\_id\_t] \*/

SAI\_PACKET\_TRAP\_ID,

/\* Ingress port [sai\_object\_id\_t] \*/

SAI\_PACKET\_INGRESS\_PORT,

/\* Ingress LAG [sai\_object\_id\_t] \*/

SAI\_PACKET\_INGRESS\_LAG,

/\* Egress port [sai\_object\_id\_t].

\* Either Port, or LAG, or both attributes, should be provided on TX \*/

SAI\_PACKET\_EGRESS\_PORT,

/\* Egress LAG [sai\_object\_id\_t] \*/

SAI\_PACKET\_EGRESS\_LAG,

/\* Egress Queue [sai\_object\_id\_t]. Optional \*/

SAI\_PACKET\_EGRESS\_QUEUE,

/\* TX type [sai\_tx\_type\_t]. Mandatory \*/

SAI\_PACKET\_TX\_TYPE,

} sai\_packet\_attr\_t;

/\*

\* Routine Description:

\* hostif receive function

\*

\* Arguments:

\* [in] hif\_id - host interface id

\* [out] buffer – packet buffer

\* [in,out] buffer\_size- [in] allocated buffer size. [out] actual packet size in bytes

\* [in,out] attr\_count – [in] allocated list size. [out] number of attributes

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

\*

\* Return Values:

\* SAI\_STATUS\_SUCCESS on success

\* SAI\_STATUS\_BUFFER\_OVERFLOW if buffer\_size is insufficient,

\* and buffer\_size will be filled with required size. Or

\* if attr\_count is insufficient, and attr\_count

\* will be filled with required count.

\* Failure status code on error

\*/

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

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

\_Out\_ void \*buffer,

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

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

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

);

/\*

\* Routine Description:

\* hostif send function

\*

\* Arguments:

\* [in] hif\_id - host interface id. only valid for send through FD channel. 0 for send

\* through CB channel.

\* [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\_object\_id\_t hif\_id,

\_Inout\_ void \*buffer,

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

\_In\_ uint32\_t attr\_count,

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

);

### Host interface API

/\*

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

\*/

typedef struct \_sai\_host\_interface\_api\_t

{

sai\_create\_host\_interface\_fn create\_host\_interface;

sai\_remove\_host\_interface\_fn remove\_host\_interface;

sai\_set\_host\_interface\_attribute\_fn set\_host\_interface\_attribute;

sai\_get\_host\_interface\_attribute\_fn get\_host\_interface\_attribute;

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\_create\_hostif\_trap\_fn create\_hostif\_trap;

sai\_remove\_hostif\_trap\_fn remove\_hostif\_trap;

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

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

sai\_create\_hostif\_user\_defined\_trap\_fn create\_hostif\_user\_defined\_trap;

sai\_remove\_hostif\_user\_defined\_trap\_fn remove\_hostif\_user\_defined\_trap;

sai\_set\_hostif\_user\_defined\_trap\_attribute\_fn set\_user\_defined\_trap\_attribute;

sai\_get\_hostif\_user\_defined\_trap\_attribute\_fn get\_user\_defined\_trap\_attribute;

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

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

} sai\_host\_interface\_api\_t;

## New Callback function for packet 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)(  \_In\_ const void \*buffer,  \_In\_ sai\_size\_t buffer\_size,  \_In\_ uint32\_t attr\_count,  \_In\_ const sai\_attribute\_t \*attr\_list  ); |
|  |

## Changes in saitypes.h

|  |
| --- |
| typedef size\_t sai\_size\_t; |
|  |

## Changes in saiswitch.h

- /\* Action for Packets with TTL 0 or 1 [sai\_packet\_action\_t]

- (default to SAI\_PACKET\_ACTION\_TRAP) \*/

- SAI\_SWITCH\_ATTR\_VIOLATION\_TTL1\_ACTION,

## Configuration example – using netdev channel

In the below example we will configure a system running L2 {STP, LACP, IGMP snooping}, L3 {BGP}.

We will create 3 QoS groups for the traffic to CPU

High – for control STP, LACP, BGP

Medium – IGMP packet, ARP

Low – L3 exception (non-resolved ARP)

The system has 4 physical ports, and the method for send and receive of CPU traffic is via OS net device

### Step 1- create three trap groups

sai\_object\_id\_t low\_prio\_group;

sai\_object\_id\_t med\_prio\_group;

sai\_object\_id\_t high\_prio\_group;

sai\_attribute\_t sai\_attr\_list[3];

sai\_attr\_list[0].id=SAI\_HOSTIF\_GROUP\_ATTR\_PRIO;

sai\_attr\_list[0].value=7;

sai\_attr\_list[1].id=SAI\_HOSTIF\_GROUP\_ATTR\_QUEUE;

sai\_attr\_list[1].value=high\_queue\_id; // high\_queue\_id is a queue element created via QoS SAI API

sai\_attr\_list[2].id= SAI\_HOSTIF\_GROUP\_ATTR\_POLICER;

sai\_attr\_list[2].value=high\_policer\_id; //high\_policer\_id is a policer element created via policer SAI API

sai\_create\_hostif\_group\_fn(&high\_prio\_group, 3, sai\_attr\_list);

sai\_attr\_list[0].id=SAI\_HOSTIF\_GROUP\_ATTR\_PRIO;

sai\_attr\_list[0].value=6;

sai\_attr\_list[1].id= SAI\_HOSTIF\_GROUP\_ATTR\_QUEUE;

sai\_attr\_list[1].value=med\_queue\_id; // high\_queue\_id is a queue element created via QoS SAI API

sai\_attr\_list[2].id= SAI\_HOSTIF\_GROUP\_ATTR\_POLICER;

sai\_attr\_list[2].value=med\_policer\_id; //high\_policer\_id is a policer element created via policer SAI API

sai\_create\_hostif\_group\_fn(&med\_prio\_group,3, sai\_attr\_list);

sai\_attr\_list[0].id=SAI\_HOSTIF\_GROUP\_ATTR\_PRIO;

sai\_attr\_list[0].value=5;

sai\_attr\_list[1].id= SAI\_HOSTIF\_GROUP\_ATTR\_QUEUE;

sai\_attr\_list[1].value=low\_queue\_id; // high\_queue\_id is a queue element created via QoS SAI API

sai\_attr\_list[2].id= SAI\_HOSTIF\_GROUP\_ATTR\_POLICER;

sai\_attr\_list[2].value=low\_policer\_id; //high\_policer\_id is a policer element created via policer SAI API

sai\_create\_hostif\_group\_fn(&low\_prio\_group,3, sai\_attr\_list);

### Step 2- create host interface channel

Creation of port net devices in the OS

sai\_object\_id\_t host\_if\_id[4];

sai\_attribute\_t sai\_if\_channel\_attr[3];

sai\_if\_channel\_attr[0].id=SAI\_HOST\_INTERFACE\_ATTR\_TYPE;

sai\_if\_channel\_attr[0].value=SAI\_HOST\_INTERFACE\_TYPE\_PORT;

sai\_if\_channel\_attr[1].id= SAI\_HOST\_INTERFACE\_TYPE\_PORT;

sai\_if\_channel\_attr[1].value=port\_id[0]; // port\_id[X] is a port element created via port SAI API

sai\_if\_channel\_attr[2].id= SAI\_HOST\_INTERFACE\_ATTR\_NAME;

sai\_if\_channel\_attr[2].value=”port1”;

sai\_create\_host\_interface\_fn(&host\_if\_id[0], 3, sai\_if\_channel\_attr);

sai\_if\_channel\_attr[0].id= SAI\_HOST\_INTERFACE\_ATTR\_TYPE;

sai\_if\_channel\_attr[0].value=SAI\_HOST\_INTERFACE\_TYPE\_PORT;

sai\_if\_channel\_attr[1].id= SAI\_HOST\_INTERFACE\_TYPE\_PORT;

sai\_if\_channel\_attr[1].value=port\_id[1]; // port\_id[X] is a port element created via port SAI API

sai\_if\_channel\_attr[2].id= SAI\_HOST\_INTERFACE\_ATTR\_NAME;

sai\_if\_channel\_attr[2].value=”port2”;

sai\_create\_host\_interface\_fn(&host\_if\_id[1] ,3, sai\_if\_channel\_attr);

sai\_if\_channel\_attr[0].id= SAI\_HOST\_INTERFACE\_ATTR\_TYPE;

sai\_if\_channel\_attr[0].value=SAI\_HOST\_INTERFACE\_TYPE\_PORT;

sai\_if\_channel\_attr[1].id= SAI\_HOST\_INTERFACE\_TYPE\_PORT;

sai\_if\_channel\_attr[1].value=port\_id[2]; // port\_id[X] is a port element created via port SAI API

sai\_if\_channel\_attr[2].id= SAI\_HOST\_INTERFACE\_ATTR\_NAME;

sai\_if\_channel\_attr[2].value=”port3”;

sai\_create\_host\_interface\_fn(&host\_if\_id[2] ,3, sai\_if\_channel\_attr);

sai\_if\_channel\_attr[0].id= SAI\_HOST\_INTERFACE\_ATTR\_TYPE;

sai\_if\_channel\_attr[0].value=SAI\_HOST\_INTERFACE\_TYPE\_PORT;

sai\_if\_channel\_attr[1].id= SAI\_HOST\_INTERFACE\_TYPE\_PORT;

sai\_if\_channel\_attr[1].value=port\_id[3]; // port\_id[X] is a port element created via port SAI API

sai\_if\_channel\_attr[2].id= SAI\_HOST\_INTERFACE\_ATTR\_NAME;

sai\_if\_channel\_attr[2].value=”port4”;

sai\_create\_host\_interface\_fn(&host\_if\_id[3], 3, sai\_if\_channel\_attr);

### Step 3- configure the trap-ids

// configure STP trap-id

sai\_attribute\_t sai\_trap\_attr[3];

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&high\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value= SAI\_PACKET\_ACTION\_TRAP*;*

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_NETDEV;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_STP, 3, sai\_trap\_attr);

// configure LACP trap-id

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&high\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value=SAI\_PACKET\_ACTION\_TRAP*;*

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_NETDEV;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_LACP, 3, sai\_trap\_attr);

// configure ARP trap-id

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&med\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value= SAI\_PACKET\_ACTION\_LOG;

sai\_trap\_attr[2].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_NETDEV;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_ARP\_REQUEST, 3, sai\_trap\_attr);

…

//configure router exception

sai\_trap\_attr[0].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&low\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value= SAI\_PACKET\_ACTION\_TRAP;

sai\_trap\_attr[2].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_NETDEV;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_ROUTER, 3, sai\_trap\_attr);

## Configuration example – using file descriptor channel

Same setup and packet types as above example working with a single file descriptor.

### Step 1- create three trap groups

Same as 2.4.1

### Step 2- create host interface FD

Creation of the file descriptor (the same logic and flow can be used in order to create more than one file descriptor)

sai\_object\_id\_t host\_if\_id;

sai\_attribute\_t sai\_if\_channel\_attr[1];

sai\_if\_channel\_attr[0].id=SAI\_HOST\_INTERFACE\_ATTR\_TYPE;

sai\_if\_channel\_attr[0].value=SAI\_HOST\_INTERFACE\_TYPE\_FD;

sai\_create\_host\_interface\_fn(&host\_if\_id, 1, sai\_if\_channel\_attr);

### Step 3- configure the trap-ids

// configure STP trap-id

sai\_attribute\_t sai\_trap\_attr[4];

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&high\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value= SAI\_PACKET\_ACTION\_TRAP*;*

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_FD;

sai\_trap\_attr[3].id=SAI\_HOSTIF\_TRAP\_ATTR\_FD;

sai\_trap\_attr[3].value=host\_if\_id;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_STP, 3, sai\_trap\_attr);

// configure LACP trap-id

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&high\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value=SAI\_PACKET\_ACTION\_TRAP*;*

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_FD;

sai\_trap\_attr[3].id=SAI\_HOSTIF\_TRAP\_ATTR\_FD;

sai\_trap\_attr[3].value=host\_if\_id;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_LACP, 3, sai\_trap\_attr);

// configure ARP trap-id

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&med\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value= SAI\_PACKET\_ACTION\_LOG;

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_FD;

sai\_trap\_attr[3].id=SAI\_HOSTIF\_TRAP\_ATTR\_FD;

sai\_trap\_attr[3].value=host\_if\_id;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_ARP\_REQUEST, 3, sai\_trap\_attr);

…

//configure router exception

sai\_trap\_attr[0].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&low\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value= SAI\_PACKET\_ACTION\_TRAP;

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_FD;

sai\_trap\_attr[3].id=SAI\_HOSTIF\_TRAP\_ATTR\_FD;

sai\_trap\_attr[3].value=host\_if\_id;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_ROUTER, 3, sai\_trap\_attr);

### Step 4- send and receive

// receive flow

void \*buffer;

sai\_size\_t buffer\_size,

sai\_attribute\_t sai\_packet\_attr[3];

uint32\_t attr\_count = 3;

// alloocate buffer

buffer\_alloc(&buffer, CONTROL\_MTU);

while(TRUE)

{

buffer\_size = CONTROL\_MTU;

sai\_recv\_hostif\_packet\_fn(host\_if\_id, buffer, &buffer\_size, &attr\_count, sai\_packet\_attr);

handle\_packet(buffer, buffer\_size, attr\_count, sai\_packet\_attr);

);

// send flow

void \*buffer;

sai\_attribute\_t sai\_packet\_send\_attr[3];

// send STP packet through port 4

buffer\_alloc(&buffer, STP\_FRAME\_SIZE);

stp\_buffer\_set(buffer);

sai\_packet\_send\_attr[0].id= SAI\_PACKET\_EGRESS\_PORT;

sai\_packet\_send\_attr[0].value=4;

sai\_packet\_send\_attr[1].id= SAI\_PACKET\_EGRESS\_QUEUE;

sai\_packet\_send\_attr[1].value= queue\_id; //queue\_id is a queue element created via QoS SAI;

sai\_packet\_send\_attr[2].id= SAI\_PACKET\_TX\_TYPE;

sai\_packet\_send\_attr[2].value= SAI\_TX\_TYPE\_PIPELINE\_BYPASS;

sai\_send\_hostif\_packet\_fn(host\_if\_id, buffer, STP\_FRAME\_SIZE, 3, sai\_packet\_send\_attr);

## Configuration example – using packet receive callback channel

Same setup and packet types as above example working with a packet receive callback

### Step 1- create three trap groups

Same as 2.4.1

### Step 2- configure the trap-ids

// configure STP trap-id

sai\_attribute\_t sai\_trap\_attr[3];

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&high\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value= SAI\_PACKET\_ACTION\_TRAP*;*

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_CB;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_STP, 3, sai\_trap\_attr);

// configure LACP trap-id

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&high\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value=SAI\_PACKET\_ACTION\_TRAP*;*

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_CB;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_LACP, 3, sai\_trap\_attr);

// configure ARP trap-id

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&med\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value= SAI\_PACKET\_ACTION\_LOG;

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_CB;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_ARP\_REQUEST, 3, sai\_trap\_attr);

…

//configure router exception

sai\_trap\_attr[0].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_GROUP;

sai\_trap\_attr[0].value=&low\_prio\_group;

sai\_trap\_attr[1].id= SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_ACTION;

sai\_trap\_attr[1].value= SAI\_PACKET\_ACTION\_TRAP;

sai\_trap\_attr[2].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[2].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_CB;

sai\_create\_hostif\_trap\_fn(SAI\_HOSTIF\_TRAP\_ID\_ROUTER, 3, sai\_trap\_attr);

### Step 4- send and receive

// recive flow SAI via CB

SAI calls application callback on\_packet\_event(

\_In\_ const void \*buffer,

\_In\_ sai\_size\_t buffer\_size,

\_In\_ uint32\_t attr\_count,

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

)

// send flow

void \* buffer;

sai\_attribute\_t sai\_packet\_send\_attr[3];

// send STP packet through port 4

buffer\_alloc(&buffer, STP\_FRAME\_SIZE);

stp\_buffer\_set(buffer);

sai\_packet\_send\_attr[0].id= SAI\_PACKET\_EGRESS\_PORT;

sai\_packet\_send\_attr[0].value=4;

sai\_packet\_send\_attr[1].id= SAI\_PACKET\_EGRESS\_QUEUE;

sai\_packet\_send\_attr[1].value= queue\_id; //queue\_id is a queue element created via QoS SAI;

sai\_packet\_send\_attr[2].id= SAI\_PACKET\_TX\_TYPE;

sai\_packet\_send\_attr[2].value= SAI\_TX\_TYPE\_PIPELINE\_BYPASS;

sai\_send\_hostif\_packet\_fn(0, buffer, STP\_FRAME\_SIZE, 3 ,sai\_packet\_send\_attr);

## Configuration example – trap via ACL

In the below example we will configure a system running using ACL to trap packet to the CPU

The user channel will be set globally ( not per trap-id ) to OS net device

### Step 1- configure the default user define trap id to use OS net device

sai\_hostif\_user\_defined\_trap\_attr\_t sai\_trap\_attr[1];

sai\_trap\_attr[0].id=SAI\_HOSTIF\_TRAP\_ATTR\_TRAP\_CHANNEL;

sai\_trap\_attr[0].value= SAI\_HOSTIF\_TRAP\_CHANNEL\_NETDEV;

sai\_create\_hostif\_user\_defined\_trap\_fn(SAI\_HOSTIF\_USER\_DEFINED\_TRAP\_ID\_DEFAULT,

1, sai\_trap\_attr);

### Step 2- ACL configuration

* Create ACL table
  + Set ACL table priority attribute
* Create an ACL entry
  + Set ACL entry match fields
  + Set ACL entry priority
    - Set ACL entry action fields
      * SAI\_ACL\_ENTRY\_ATTR\_ACTION\_TRAP\_TO\_CPU to true
      * SAI\_ACL\_ENTRY\_ATTR\_ACTION\_SET\_POLICER with pre define policer
      * SAI\_ACL\_ENTRY\_ATTR\_ACTION\_SET\_COS with pre define traffic class
  + Bing ACL entry to a ACL table