

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **Bidirectional Forwarding Detection (BFD)** |
| **Authors** | **DELL Networking** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **09/25/2017** |
| **SAI-Version** | **V0.3** |

1 Overview 3

1.1 Protocol overview and Session establishment 3

1.2 BFD Session modes 4

1.3 Poll Sequence 4

1.4 Demand Sequence 4

1.5 BFD state machine 4

1.6 BFD HW Offload 5

2 Specification 5

2.1 saibfd.h (new file) 5

2.2 Changes to saiswitch.h 15

3 Examples 17

3.1 Create BFD Session 17

3.1.1 Create using Endpoint ip address 17

3.1.2 Choosing a path between the endpoints 18

3.2 Remove BFD Session 20

# Overview

Traditionally the L3 routing protocols such as OSPF, BGP etc., use the slow Hello mechanisms to detect the failures which are no more than a second. This detection time is too long for applications and in turn result in great deal of data loss.

Bidirectional Forwarding Detection (BFD) provides low over-head, shorter detection time of failures in the path between two adjacent forwarding engines.

The BFD state machine implements a three-way handshake, both when establishing a BFD session and when tearing it down for any reason, to ensure that both systems are aware of the state change.

BFD can be abstracted as a simple service. The service primitives provided by BFD are to create, destroy, and modify a session, given the destination address and other parameters. BFD in return provides a signal to its clients indicating when the BFD session goes up or down.

## Protocol overview and Session establishment

BFD is a simple Hello protocol, similar to that most of the L3 routing protocols. BFD runs between a pair of system exchanging BFD packets periodically.

BFD control Packet

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|Vers | Diag |Sta|P|F|C|A|D|M| Detect Mult | Length |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| My Discriminator |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Your Discriminator |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Desired Min TX Interval |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Required Min RX Interval |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Required Min Echo RX Interval |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

An optional Authentication Section MAY be present:

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Auth Type | Auth Len | Authentication Data... |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

BFD uses two-way communication establishment a.k.a three packet handshake to declare the session is operational. During the initial session establishment, the BFD packet interval are negotiated. However these accepted frequency can be dynamically modified in real time.

## BFD Session modes

There are two modes where BFD can operate. And the system can be either Active or Passive.

Asynchronous: In this mode, both the systems send the BFD control packet to one another periodically. If a packet is not received from the other system for a long enough duration then the session is declared down

Demand Mode: As the name suggests there is no periodic packet exchange happens in this mode. But when the system feels to verify the connectivity a short sequence of BFD control packets are exchanged to verify connectivity. This is to reduce overhead.

Echo Function: Parallel to these modes, BFD support the Echo function, where the system generates a stream of bfd echo packets which is looped back by the other system. If number of those packets are not received then the session is declared down.

## Poll Sequence

The Poll sequence is, one system sets the poll bit in the BFD control packets and the other system respond with the Final bit set, which implies that the other system has seen the poll packet. There are two use cases for this sequence. One is to inform any parameter changes with respect to that BFD session and the second is to verify connectivity in Demand mode.

## Demand Sequence

Demand mode is requested independently in each direction by virtue of a system setting the Demand (D) bit in its BFD Control packets. The system receiving the Demand bit ceases the periodic transmission of BFD Control packets. If both systems are operating in Demand mode, no periodic BFD Control packets will flow in either direction.

When a system in Demand mode wants to verify connectivity, it initiates the BFD control packets with Poll bit set. If it does not receive any packet with Final bit set for that session within the expiration time the session will be declared down.

## BFD state machine

The BFD state machine is very simple and straight forward. The BFD state machine is driven by the state in the received BFD packet in combination with local state.

Below diagram explains the BFD state machine

+--+

| | UP, ADMIN DOWN, TIMER

| V

DOWN +------+ INIT

DOWN

+------------| |------------+

| | DOWN | |

| +-------->| | <-------+ |

| | +------+ | |

| | | |

| | ADMIN DOWN,| |

| |ADMIN DOWN, DOWN,| |

| |TIMER TIMER| |

V | | V

+------+ +------+

UP

INIT

+----| | | |----+

DOWN| | INIT |--------------------->| UP | |INIT, UP

+--->| | INIT, UP | |<---+

+------+ +------+

## BFD HW Offload

Typically when the BFD session begins, the systems operate at periodic slow exchange of control packets. When the bidirectional communication is achieved the BFD session become UP.

At this point the system can choose to operate at a higher desired rate. This higher rate negotiation happen via the Poll and Final sequence explained above. Till this point the BFD session is controlled in the control plane application.

At this point the BFD application can choose to offload the session to the HW (if supported) or to an FPGA which can periodically send these BFD packets at faster rate.

This session state is further tracked by HW and any state change events are informed to the BFD control plane application for remedy.

# Specification

## saibfd.h (new file)

/\*\*

\* @brief SAI session type of BFD

\*/

typedef enum \_sai\_bfd\_session\_type\_t

{

/\*\* Demand Active Mode \*/

SAI\_BFD\_SESSION\_TYPE\_DEMAND\_ACTIVE = 0,

/\*\* Demand Passive Mode \*/

SAI\_BFD\_SESSION\_TYPE\_DEMAND\_PASSIVE,

/\*\* Asynchronous Active Mode \*/

SAI\_BFD\_SESSION\_TYPE\_ASYNC\_ACTIVE,

/\*\* Asynchronous Passive Mode \*/

SAI\_BFD\_SESSION\_TYPE\_ASYNC\_PASSIVE,

} sai\_bfd\_session\_type\_t;

/\*\*

\* @brief SAI type of encapsulation for BFD

\*/

typedef enum \_sai\_bfd\_encapsulation\_type\_t

{

/\*\*

\* @brief IP in IP Encapsulation | L2 Ethernet header | IP header | Inner IP header | Original BFD packet

\*/

SAI\_BFD\_ENCAPSULATION\_TYPE\_IP\_IN\_IP,

/\*\*

\* @brief L3 GRE Tunnel Encapsulation | L2 Ethernet header | IP header | GRE header | Original BFD packet

\*/

SAI\_BFD\_ENCAPSULATION\_TYPE\_L3\_GRE\_TUNNEL,

} sai\_bfd\_encapsulation\_type\_t;

/\*\*

\* @brief SAI BFD session state

\*/

typedef enum \_sai\_bfd\_session\_state\_t

{

/\*\* BFD Session is in Admin down \*/

SAI\_BFD\_SESSION\_STATE\_ADMIN\_DOWN,

/\*\* BFD Session is Down \*/

SAI\_BFD\_SESSION\_STATE\_DOWN,

/\*\* BFD Session is in Initialization \*/

SAI\_BFD\_SESSION\_STATE\_INIT,

/\*\* BFD Session is Up \*/

SAI\_BFD\_SESSION\_STATE\_UP,

} sai\_bfd\_session\_state\_t;

/\*\*

\* @brief Defines the operational status of the BFD session

\*/

typedef struct \_sai\_bfd\_session\_state\_notification\_t

{

/\*\* BFD Session id \*/

sai\_object\_id\_t bfd\_session\_id;

/\*\* BFD session state \*/

sai\_bfd\_session\_state\_t session\_state;

} sai\_bfd\_session\_state\_notification\_t;

/\*\*

\* @brief SAI attributes for BFD session

\*/

typedef enum \_sai\_bfd\_session\_attr\_t

{

/\*\*

\* @brief Start of attributes

\*/

SAI\_BFD\_SESSION\_ATTR\_START,

/\*\*

\* @brief BFD Session type DEMAND/ASYNCHRONOUS

\*

\* @type sai\_bfd\_session\_type\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_TYPE = SAI\_BFD\_SESSION\_ATTR\_START,

/\*\*

\* @brief Hardware lookup valid

\*

\* @type bool

\* @flags CREATE\_ONLY

\* @default false

\*/

SAI\_BFD\_SESSION\_ATTR\_HW\_LOOKUP\_VALID,

/\*\*

\* @brief Virtual Router

\*

\* @type sai\_object\_id\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET

\* @objects SAI\_OBJECT\_TYPE\_VIRTUAL\_ROUTER

\* @condition SAI\_BFD\_SESSION\_ATTR\_HW\_LOOKUP\_VALID == true

\*/

SAI\_BFD\_SESSION\_ATTR\_VIRTUAL\_ROUTER,

/\*\*

\* @brief Destination Port

\*

\* @type sai\_object\_id\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET

\* @objects SAI\_OBJECT\_TYPE\_PORT

\* @condition SAI\_BFD\_SESSION\_ATTR\_HW\_LOOKUP\_VALID == false

\*/

SAI\_BFD\_SESSION\_ATTR\_PORT,

/\*\*

\* @brief Local discriminator

\*

\* @type sai\_uint32\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_LOCAL\_DISCRIMINATOR,

/\*\*

\* @brief Remote discriminator

\*

\* @type sai\_uint32\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_REMOTE\_DISCRIMINATOR,

/\*\*

\* @brief UDP Source port

\*

\* @type sai\_uint32\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_UDP\_SRC\_PORT,

/\*\*

\* @brief Class-of-Service (Traffic Class)

\*

\* @type sai\_uint8\_t

\* @flags CREATE\_AND\_SET

\* @default 0

\*/

SAI\_BFD\_SESSION\_ATTR\_TC,

/\*\*

\* @brief L2 header TPID.

\*

\* @type sai\_uint16\_t

\* @flags CREATE\_AND\_SET

\* @isvlan false

\* @default 0x8100

\* @validonly SAI\_BFD\_SESSION\_ATTR\_VLAN\_HEADER\_VALID == true

\*/

SAI\_BFD\_SESSION\_ATTR\_VLAN\_TPID,

/\*\*

\* @brief L2 header VLAN Id.

\*

\* @type sai\_uint16\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\* @isvlan true

\* @validonly SAI\_BFD\_SESSION\_ATTR\_VLAN\_HEADER\_VALID == true

\*/

SAI\_BFD\_SESSION\_ATTR\_VLAN\_ID,

/\*\*

\* @brief L2 header packet priority (3 bits).

\*

\* @type sai\_uint8\_t

\* @flags CREATE\_AND\_SET

\* @default 0

\* @validonly SAI\_BFD\_SESSION\_ATTR\_VLAN\_HEADER\_VALID == true

\*/

SAI\_BFD\_SESSION\_ATTR\_VLAN\_PRI,

/\*\*

\* @brief L2 header Vlan CFI (1 bit).

\*

\* @type sai\_uint8\_t

\* @flags CREATE\_AND\_SET

\* @default 0

\* @validonly SAI\_BFD\_SESSION\_ATTR\_VLAN\_HEADER\_VALID == true

\*/

SAI\_BFD\_SESSION\_ATTR\_VLAN\_CFI,

/\*\*

\* @brief Vlan header valid

\*

\* @type bool

\* @flags CREATE\_ONLY

\* @default false

\* @validonly SAI\_BFD\_SESSION\_ATTR\_HW\_LOOKUP\_VALID == false

\*/

SAI\_BFD\_SESSION\_ATTR\_VLAN\_HEADER\_VALID,

/\*\*

\* @brief Encapsulation type

\*

\* @type sai\_bfd\_encapsulation\_type\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\* @validonly SAI\_BFD\_SESSION\_ATTR\_HW\_LOOKUP\_VALID == false

\*/

SAI\_BFD\_SESSION\_ATTR\_BFD\_ENCAPSULATION\_TYPE,

/\*\*

\* @brief IP header version

\*

\* @type sai\_uint8\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET

\*/

SAI\_BFD\_SESSION\_ATTR\_IPHDR\_VERSION,

/\*\*

\* @brief IP header TOS

\*

\* @type sai\_uint8\_t

\* @flags CREATE\_AND\_SET

\* @default 0

\*/

SAI\_BFD\_SESSION\_ATTR\_TOS,

/\*\*

\* @brief IP header TTL

\*

\* @type sai\_uint8\_t

\* @flags CREATE\_AND\_SET

\* @default 255

\*/

SAI\_BFD\_SESSION\_ATTR\_TTL,

/\*\*

\* @brief Source IP

\*

\* @type sai\_ip\_address\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_SRC\_IP\_ADDRESS,

/\*\*

\* @brief Destination IP

\*

\* @type sai\_ip\_address\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_DST\_IP\_ADDRESS,

/\*\*

\* @brief Tunnel IP header TOS

\*

\* @type sai\_uint8\_t

\* @flags CREATE\_AND\_SET

\* @default 0

\* @validonly SAI\_BFD\_SESSION\_ATTR\_BFD\_ENCAPSULATION\_TYPE == SAI\_BFD\_ENCAPSULATION\_TYPE\_IP\_IN\_IP

\*/

SAI\_BFD\_SESSION\_ATTR\_TUNNEL\_TOS,

/\*\*

\* @brief Tunnel IP header TTL

\*

\* @type sai\_uint8\_t

\* @flags CREATE\_AND\_SET

\* @default 255

\* @validonly SAI\_BFD\_SESSION\_ATTR\_BFD\_ENCAPSULATION\_TYPE == SAI\_BFD\_ENCAPSULATION\_TYPE\_IP\_IN\_IP

\*/

SAI\_BFD\_SESSION\_ATTR\_TUNNEL\_TTL,

/\*\*

\* @brief Tunnel source IP

\*

\* @type sai\_ip\_address\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\* @condition SAI\_BFD\_SESSION\_ATTR\_BFD\_ENCAPSULATION\_TYPE == SAI\_BFD\_ENCAPSULATION\_TYPE\_IP\_IN\_IP

\*/

SAI\_BFD\_SESSION\_ATTR\_TUNNEL\_SRC\_IP\_ADDRESS,

/\*\*

\* @brief Tunnel destination IP

\*

\* @type sai\_ip\_address\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY

\* @condition SAI\_BFD\_SESSION\_ATTR\_BFD\_ENCAPSULATION\_TYPE == SAI\_BFD\_ENCAPSULATION\_TYPE\_IP\_IN\_IP

\*/

SAI\_BFD\_SESSION\_ATTR\_TUNNEL\_DST\_IP\_ADDRESS,

/\*\*

\* @brief L2 source MAC address

\*

\* @type sai\_mac\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET

\* @condition SAI\_BFD\_SESSION\_ATTR\_HW\_LOOKUP\_VALID == false

\*/

SAI\_BFD\_SESSION\_ATTR\_SRC\_MAC\_ADDRESS,

/\*\*

\* @brief L2 destination MAC address

\*

\* @type sai\_mac\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET

\* @condition SAI\_BFD\_SESSION\_ATTR\_HW\_LOOKUP\_VALID == false

\*/

SAI\_BFD\_SESSION\_ATTR\_DST\_MAC\_ADDRESS,

/\*\*

\* @brief To enable echo function on BFD session

\*

\* @type bool

\* @flags CREATE\_AND\_SET

\* @default false

\*/

SAI\_BFD\_SESSION\_ATTR\_ECHO\_ENABLE,

/\*\*

\* @brief Multi hop BFD session

\*

\* @type bool

\* @flags CREATE\_ONLY

\* @default false

\*/

SAI\_BFD\_SESSION\_ATTR\_MULTIHOP,

/\*\*

\* @brief Control Plane Independent

\*

\* @type bool

\* @flags CREATE\_ONLY

\* @default false

\*/

SAI\_BFD\_SESSION\_ATTR\_CBIT,

/\*\*

\* @brief Minimum Transmit interval in microseconds

\*

\* @type sai\_uint32\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET

\*/

SAI\_BFD\_SESSION\_ATTR\_MIN\_TX,

/\*\*

\* @brief Minimum Receive interval in microseconds

\*

\* @type sai\_uint32\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET

\*/

SAI\_BFD\_SESSION\_ATTR\_MIN\_RX,

/\*\*

\* @brief Detect time Multiplier

\*

\* @type sai\_uint8\_t

\* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET

\*/

SAI\_BFD\_SESSION\_ATTR\_MULTIPLIER,

/\*\*

\* @brief Minimum Remote Transmit interval in microseconds

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_REMOTE\_MIN\_TX,

/\*\*

\* @brief Minimum Remote Receive interval in microseconds

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_REMOTE\_MIN\_RX,

/\*\*

\* @brief BFD Session state

\*

\* @type sai\_bfd\_session\_state\_t

\* @flags READ\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_STATE,

/\*\*

\* @brief End of attributes

\*/

SAI\_BFD\_SESSION\_ATTR\_END,

/\*\* Custom range base value \*/

SAI\_BFD\_SESSION\_ATTR\_CUSTOM\_RANGE\_START = 0x10000000,

/\*\* End of custom range base \*/

SAI\_BFD\_SESSION\_ATTR\_CUSTOM\_RANGE\_END

} sai\_bfd\_session\_attr\_t;

/\*\*

\* @brief BFD Session counter IDs in sai\_get\_bfd\_session\_stats() call

\*/

typedef enum \_sai\_bfd\_session\_stat\_t

{

/\*\* Ingress packet stat count \*/

SAI\_BFD\_SESSION\_STAT\_IN\_PACKETS,

/\*\* Egress packet stat count \*/

SAI\_BFD\_SESSION\_STAT\_OUT\_PACKETS,

/\*\* Packet Drop stat count \*/

SAI\_BFD\_SESSION\_STAT\_DROP\_PACKETS

} sai\_bfd\_session\_stat\_t;

/\*\*

\* @brief Create BFD session.

\*

\* @param[out] bfd\_session\_id BFD session id

\* @param[in] switch\_id Switch id

\* @param[in] attr\_count Number of attributes

\* @param[in] attr\_list Value of attributes

\*

\* @return #SAI\_STATUS\_SUCCESS if operation is successful otherwise a different

\* error code is returned.

\*/

typedef sai\_status\_t (\*sai\_create\_bfd\_session\_fn)(

\_Out\_ sai\_object\_id\_t \*bfd\_session\_id,

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

\_In\_ uint32\_t attr\_count,

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

/\*\*

\* @brief Remove BFD session.

\*

\* @param[in] bfd\_session\_id BFD session id

\*

\* @return #SAI\_STATUS\_SUCCESS if operation is successful otherwise a different

\* error code is returned.

\*/

typedef sai\_status\_t (\*sai\_remove\_bfd\_session\_fn)(

\_In\_ sai\_object\_id\_t bfd\_session\_id);

/\*\*

\* @brief Set BFD session attributes.

\*

\* @param[in] bfd\_session\_id BFD session id

\* @param[in] attr Value of attribute

\*

\* @return #SAI\_STATUS\_SUCCESS if operation is successful otherwise a different

\* error code is returned.

\*/

typedef sai\_status\_t (\*sai\_set\_bfd\_session\_attribute\_fn)(

\_In\_ sai\_object\_id\_t bfd\_session\_id,

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

/\*\*

\* @brief Get BFD session attributes.

\*

\* @param[in] bfd\_session\_id BFD session id

\* @param[in] attr\_count Number of attributes

\* @param[inout] attr\_list Value of attribute

\*

\* @return #SAI\_STATUS\_SUCCESS if operation is successful otherwise a different

\* error code is returned.

\*/

typedef sai\_status\_t (\*sai\_get\_bfd\_session\_attribute\_fn)(

\_In\_ sai\_object\_id\_t bfd\_session\_id,

\_In\_ uint32\_t attr\_count,

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

/\*\*

\* @brief Get BFD session statistics counters.

\*

\* @param[in] bfd\_session\_id BFD session id

\* @param[in] number\_of\_counters Number of counters in the array

\* @param[in] counter\_ids Specifies the array of counter ids

\* @param[out] counters Array of resulting counter values.

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_get\_bfd\_session\_stats\_fn)(

\_In\_ sai\_object\_id\_t bfd\_session\_id,

\_In\_ uint32\_t number\_of\_counters,

\_In\_ const sai\_bfd\_session\_stat\_t \*counter\_ids,

\_Out\_ uint64\_t \*counters);

/\*\*

\* @brief Clear BFD session statistics counters.

\*

\* @param[in] bfd\_session\_id BFD session id

\* @param[in] number\_of\_counters Number of counters in the array

\* @param[in] counter\_ids Specifies the array of counter ids

\*

\* @return #SAI\_STATUS\_SUCCESS on success, failure status code on error

\*/

typedef sai\_status\_t (\*sai\_clear\_bfd\_session\_stats\_fn)(

\_In\_ sai\_object\_id\_t bfd\_session\_id,

\_In\_ uint32\_t number\_of\_counters,

\_In\_ const sai\_bfd\_session\_stat\_t \*counter\_ids);

/\*\*

\* @brief BFD session state change notification

\*

\* Passed as a parameter into sai\_initialize\_switch()

\*

\* @count data[count]

\*

\* @param[in] count Number of notifications

\* @param[in] data Array of BFD session state

\*/

typedef void (\*sai\_bfd\_session\_state\_change\_notification\_fn)(

\_In\_ uint32\_t count,

\_In\_ sai\_bfd\_session\_state\_notification\_t \*data);

/\*\*

\* @brief BFD method table retrieved with sai\_api\_query()

\*/

typedef struct \_sai\_bfd\_api\_t

{

sai\_create\_bfd\_session\_fn create\_bfd\_session;

sai\_remove\_bfd\_session\_fn remove\_bfd\_session;

sai\_set\_bfd\_session\_attribute\_fn set\_bfd\_session\_attribute;

sai\_get\_bfd\_session\_attribute\_fn get\_bfd\_session\_attribute;

sai\_get\_bfd\_session\_stats\_fn get\_bfd\_session\_stats;

sai\_clear\_bfd\_session\_stats\_fn clear\_bfd\_session\_stats;

} sai\_bfd\_api\_t;

## Changes to saiswitch.h

typedef enum \_sai\_switch\_attr\_t

{

……

………

/\*\*

\* @brief Set Switch BFD session state change event notification callback function passed to the adapter.

\*

\* Use sai\_bfd\_session\_state\_change\_notification\_fn as notification function.

\*

\* @type sai\_pointer\_t sai\_bfd\_session\_state\_change\_notification\_fn

\* @flags CREATE\_AND\_SET

\* @default NULL

\*/

SAI\_SWITCH\_ATTR\_BFD\_SESSION\_STATE\_NOTIFY,

/\*\*

\* @brief Number of BFD session in the NPU

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_NUMBER\_OF\_BFD\_SESSION,

/\*\*

\* @brief Max number of BFD session NPU supports

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_MAX\_BFD\_SESSION,

/\*\*

\* @brief Minimum Receive interval NPU supports in microseconds

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_MIN\_BFD\_RX,

/\*\*

\* @brief Minimum Transmit interval NPU supports in microseconds

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_MIN\_BFD\_TX,

/\*\*

\* @brief End of attributes \*/

SAI\_SWITCH\_ATTR\_END,

……..

} sai\_switch\_attr\_t;

# Examples

## Create BFD Session

### Create using Endpoint ip address

A BFD session between two endpoints can be created by simply specifying the bfd ip packet attributes. In this case, either SAI or NPU can resolve the encap and egress interface.

sai\_api\_query(SAI\_API\_BFD, &bfd\_api);

sai\_object\_id\_t bfd\_session = 0;

sai\_attribute\_t bfd\_attr[] = {0};

bfd\_attr[0].id = SAI\_BFD\_SESSION\_ATTR\_TYPE;

bfd\_attr[0].value.s32 = SAI\_BFD\_SESSION\_TYPE\_ASYNC\_ACTIVE;

bfd\_attr[1].id = SAI\_BFD\_SESSION\_ATTR\_VIRTUAL\_ROUTER;

bfd\_attr[1].value.oid = VRF\_object\_id;

bfd\_attr[2].id = SAI\_BFD\_SESSION\_ATTR\_LOCAL\_DISCRIMINATOR;

bfd\_attr[2].value.s32 = 2893;

bfd\_attr[3].id = SAI\_BFD\_SESSION\_ATTR\_REMOTE\_DISCRIMINATOR;

bfd\_attr[3].value.s32 = 9345;

bfd\_attr[4].id = SAI\_BFD\_SESSION\_ATTR\_UDP\_SRC\_PORT;

bfd\_attr[4].value.s32 = 8499;

bfd\_attr[8].id = SAI\_BFD\_SESSION\_ATTR\_IPHDR\_VERSION;

bfd\_attr[8].value.u8 = ipv4;

bfd\_attr[9].id = SAI\_BFD\_SESSION\_ATTR\_MULTIPLIER;

bfd\_attr[9].value.s32 = 5;

bfd\_attr[10].id =SAI\_BFD\_SESSION\_ATTR\_SRC\_IP\_ADDRESS;

bfd\_attr[10].value.ipaddr.addr.ip4 = 0xa0101002;

bfd\_attr[10].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

bfd\_attr[11].id =SAI\_BFD\_SESSION\_ATTR\_DST\_IP\_ADDRESS;

bfd\_attr[11].value.ipaddr.addr.ip4 = 0xa0101011;

bfd\_attr[11].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

bfd\_attr[12].id = SAI\_BFD\_SESSION\_ATTR\_MIN\_TX;

bfd\_attr[12].value.s32 = 500;

bfd\_attr[13].id = SAI\_BFD\_SESSION\_ATTR\_MIN\_RX;

bfd\_attr[13].value.s32 = 500;

bfd\_api-> create\_bfd\_session(&bfd\_session, 15, bfd\_attr);

### Choosing a path between the endpoints

On the other hand, the RFC states that there can be Multiple BFD sessions established between pair of two systems over multiple path between those systems. Below is mentioned in the rfc5880

“Multiple BFD sessions can be established between the same

pair of systems when multiple paths between them are present in at

least one direction, even if a lesser number of paths are available

in the other direction“

In this case SAI expects the application to tell the attributes related to encapsulation and egress physical interface.

The other case being running micro bfd sessions on each lag member link instead of running bfd session on the lag interface itself.

sai\_api\_query(SAI\_API\_BFD, &bfd\_api);

sai\_object\_id\_t bfd\_session = 0;

sai\_attribute\_t bfd\_attr[] = {0};

bfd\_attr[0].id = SAI\_BFD\_SESSION\_ATTR\_TYPE;

bfd\_attr[0].value.s32 = SAI\_BFD\_SESSION\_TYPE\_ASYNC\_ACTIVE;

bfd\_attr[1].id = SAI\_BFD\_SESSION\_ATTR\_HW\_LOOKUP\_VALID;

bfd\_attr[1].value.oid = false;

bfd\_attr[2].id = SAI\_BFD\_SESSION\_ATTR\_PORT;

bfd\_attr[2].value.oid = port\_object\_id;

bfd\_attr[3].id = SAI\_BFD\_SESSION\_ATTR\_LOCAL\_DISCRIMINATOR;

bfd\_attr[3].value.s32 = 2893;

bfd\_attr[4].id = SAI\_BFD\_SESSION\_ATTR\_REMOTE\_DISCRIMINATOR;

bfd\_attr[4].value.s32 = 9345;

bfd\_attr[5].id = SAI\_BFD\_SESSION\_ATTR\_UDP\_SRC\_PORT;

bfd\_attr[5].value.s32 = 8499;

bfd\_attr[6].id = SAI\_BFD\_SESSION\_ATTR\_VLAN\_HEADER\_VALID;

bfd\_attr[6].value.booldata=true;

bfd\_attr[7].id = SAI\_BFD\_SESSION\_ATTR\_VLAN\_ID;

bfd\_attr[7].value.s32 = 100;

bfd\_attr[8].id = SAI\_BFD\_SESSION\_ATTR\_IPHDR\_VERSION;

bfd\_attr[8].value.u8 = ipv4;

bfd\_attr[9].id = SAI\_BFD\_SESSION\_ATTR\_MULTIPLIER;

bfd\_attr[9].value.s32 = 5;

bfd\_attr[10].id =SAI\_BFD\_SESSION\_ATTR\_SRC\_IP\_ADDRESS;

bfd\_attr[10].value.ipaddr.addr.ip4 = 0xa0101002;

bfd\_attr[10].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

bfd\_attr[11].id =SAI\_BFD\_SESSION\_ATTR\_DST\_IP\_ADDRESS;

bfd\_attr[11].value.ipaddr.addr.ip4 = 0xa0101011;

bfd\_attr[11].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

bfd\_attr[12].id =SAI\_BFD\_SESSION\_ATTR\_SRC\_MAC\_ADDRESS;

bfd\_attr[12].value.mac[0] = 0x00;

bfd\_attr[12].value.mac[1] = 0x01;

bfd\_attr[12].value.mac[2] = 0x02;

bfd\_attr[12].value.mac[3] = 0x03;

bfd\_attr[12].value.mac[4] = 0x04;

bfd\_attr[12].value.mac[5] = 0x05;

bfd\_attr[13].id =SAI\_BFD\_SESSION\_ATTR\_DST\_MAC\_ADDRESS;

bfd\_attr[13].value.mac[0] = 0x00;

bfd\_attr[13].value.mac[1] = 0x11;

bfd\_attr[13].value.mac[2] = 0x12;

bfd\_attr[13].value.mac[3] = 0x13;

bfd\_attr[13].value.mac[4] = 0x14;

bfd\_attr[13].value.mac[5] = 0x15;

bfd\_attr[14].id = SAI\_BFD\_SESSION\_ATTR\_MIN\_TX;

bfd\_attr[14].value.s32 = 500;

bfd\_attr[15].id = SAI\_BFD\_SESSION\_ATTR\_MIN\_RX;

bfd\_attr[15].value.s32 = 500;

bfd\_api-> create\_bfd\_session(&bfd\_session, 15, bfd\_attr);

## Remove BFD Session

bfd\_api->remove\_bfd\_session(bfd\_session\_id);