

|  |
| --- |
|  |

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.

## BFD Process



Typically BFD process is as follow:

RX：

1. Identify

Identify the BFD packet by data plane, so it need to associate BFD session to data forward entry, such as route entry

1. Lookup

Lookup the BFD session by discriminator in the BFD packet

TX：

1. Generate BFD

For IP/IPinIP/GRE/VxLAN/MPLS/PW VCCV with IP BFD, assemble IP header and BFD PDU

For PW VCCV without IP, assemble BFD PDU

1. Encapsulation

Encapsulate outer packet format by next hop, such as ipuc next hop, tunnel next hop and so on. And the next hop is just the data plane’s next hop, because a BFD session is detecting the data plane.



In addition, it can encapsulate the BFD packet more flexible by next hop. Such as:

IP(rfc5881):

L2 | IP | UDP | BFD

MPLS(rfc5884):

L2 | LSP | IP | UDP | BFD

PW VCCV(rfc5885):

L2 | LSP | PW | ACH | BFD

L2 | LSP | PW | ACH | IP | BFD

L2 | Router Alert Label | PW | ACH | IP | BFD

L2 | LSP | PW TTL=1 | ACH | IP | BFD

VxLan(draft-ietf-bfd-vxlan-00):

L2 | IP | UDP | VxLAN | Inner L2 | Inner IP | Inner UDP | BFD

# 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 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 SAI BFD diag

\*/

typedef enum \_sai\_bfd\_session\_diag\_t

{

/\*\* BFD No Diagnostic \*/

SAI\_BFD\_SESSION\_DIAG\_NONE = 0,

/\*\* BFD Control Detection Time Expired \*/

SAI\_BFD\_SESSION\_DIAG\_TIME\_EXPIRED = 1,

/\*\* BFD Echo Function Failed \*/

SAI\_BFD\_SESSION\_DIAG\_ECHO\_FAIL = 2,

/\*\* BFD Neighbor Signaled Session Down \*/

SAI\_BFD\_SESSION\_DIAG\_NEIGHBOR\_DOWN = 3,

/\*\* BFD Forwarding Plane Reset \*/

SAI\_BFD\_SESSION\_DIAG\_FORWARDING\_RESET = 4,

/\*\* BFD Path Down \*/

SAI\_BFD\_SESSION\_DIAG\_PATH\_DOWN = 5,

/\*\* BFD Concatenated Path Down \*/

SAI\_BFD\_SESSION\_DIAG\_CONCATENTED\_PATH\_DOWN = 6,

/\*\* BFD Administratively Down \*/

SAI\_BFD\_SESSION\_DIAG\_ADMINISTRATIVELY\_DOWN = 7,

/\*\* BFD Reverse Concatenated Path Down \*/

SAI\_BFD\_SESSION\_DIAG\_REVERSE\_CONCATENTED\_PATH\_DOWN = 8,

}sai\_bfd\_session\_diag\_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 true

\*/

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 For BFD transmit, include destination port and Encapsulation format

\*

\* @type sai\_object\_id\_t

\* @flags CREATE\_AND\_SET

\*/

SAI\_BFD\_SESSION\_ATTR\_NEXT\_HOP\_ID,

/\*\*

\* @brief Local diag

\*

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

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

\*/

SAI\_BFD\_SESSION\_ATTR\_LOCAL\_DIAG,

/\*\*

\* @brief Remote diag

\*

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

\* @flags READ\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_REMOTE\_DIAG,

/\*\*

\* @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 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 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(Desired Min TX Interval),that the local

system would like to use when transmitting BFD Control packets.

\*

\* @type sai\_uint32\_t

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

\*/

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

/\*\*

\* @brief Minimum Receive interval in microseconds(Required Min RX Interval),that the local

system would like to use when transmitting BFD Control packets.

\*

\* @type sai\_uint32\_t

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

\*/

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

/\*\*

\* @brief The desired Detection Time multiplier for BFD Control packets on the local system

\*

\* @type sai\_uint8\_t

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

\*/

SAI\_BFD\_SESSION\_ATTR\_MULTIPLIER,

/\*\*

\* @brief Detect time Multiplier,the value of Detect Mult received from the remote system

\*

\* @type sai\_uint8\_t

\* @flags READ\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_REMOTE\_MULTIPLIER,

/\*\*

\* @brief Minimum Remote Transmit interval in microseconds,that the remote

system would like to use when transmitting BFD Control packets.

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

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

/\*\*

\* @brief Minimum Remote Receive interval in microseconds,that the remote

system would like to use when transmitting BFD Control packets.

\*

\* @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 Remote Session State

\*

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

\* @flags READ\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_REMOTE\_STATE,

/\*\*

\* @brief Actual Transmit interval in microseconds

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_ACTUAL\_TX,

/\*\*

\* @brief Actual Receive interval in microseconds

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_BFD\_SESSION\_ATTR\_ACTUAL\_RX,

/\*\*

\* @brief Enable transmit BFD packet periodic in Asynchronous mode

\*

\* @type bool

\* @flags CREATE\_AND\_SET

\* @default false

\*/

SAI\_BFD\_SESSION\_ATTR\_TX\_ENABLE,

/\*\*

\* @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;

## 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\_CHANGE\_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 Timer negotiation supported by switch

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_BFD\_SUPPORTED\_TIMER\_NEGOTIATION,

/\*\*

\* @brief Max number of intervals in timer negotiation supported by switch

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

SAI\_SWITCH\_ATTR\_BFD\_MAX\_NUMBER\_OF\_INTRERVALS

/\*\*

\* @brief rx interval list in timer negotiation

\*

\* @type sai\_object\_list\_t

\* @objects sai\_uint32\_t

\* @flags CREATE\_AND\_SET

\*/

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

/\*\*

\* @brief tx interval list in timer negotiation

\*

\* @type sai\_object\_list\_t

\* @objects sai\_uint32\_t

\* @flags CREATE\_AND\_SET

\*/

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

} sai\_switch\_attr\_t;

## Changes to saihostif.h

typedef enum \_sai\_hostif\_trap\_type\_t

{

……

/\*\*

\* @brief Packets trapped when first BFD packet(Received BFD packet's Your Discriminator with 0) arrive at local system.

\* (default packet action is drop)

\*/

SAI\_HOSTIF\_TRAP\_TYPE\_LEARNING\_BFD = 0x00004005,

/\*\*

\* @brief Packets trapped when received BFD packet with P/F set arrive at local system.

\* (default packet action is drop)

\*/

SAI\_HOSTIF\_TRAP\_TYPE\_BFD\_TIMER\_NEGOTIATION = 0x00004006,

…….

}

# Examples



As shown, a pair of MEPs are deployed on R1 and R2

## Create BFD Session

### R1 configuration

//1. Create rif on port1

sai\_api\_query(SAI\_API\_ROUTER\_INTERFACE, &rif\_api);

sai\_object\_id\_t rif\_id = 0;

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

rif\_attr[0].id = SAI\_ROUTER\_INTERFACE\_ATTR\_TYPE;

rif\_attr[0].value.s32 = SAI\_ROUTER\_INTERFACE\_TYPE\_PORT;

rif\_attr[1].id = SAI\_ROUTER\_INTERFACE\_ATTR\_PORT\_ID;

rif\_attr[1].value.oid = port1;

rif\_api->create\_router\_interface(&rif\_id, switch\_id, 2, rif\_attr);

//2. Create next hop for bfd outgoing, and ip address is R2

sai\_object\_id\_t nexthop\_id = 0;

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

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

nexthop\_attr[0].value.s32 = SAI\_NEXT\_HOP\_TYPE\_IP;

nexthop\_attr[1].id = SAI\_NEXT\_HOP\_ATTR\_IP;

nexthop\_attr[1].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

nexthop\_attr[1].value.ipaddr.ip4 = 0x01010102;

nexthop\_attr[2].id = SAI\_NEXT\_HOP\_ATTR\_ROUTER\_INTERFACE\_ID;

nexthop\_attr[2].value.oid = rif\_id;

nexthop\_api->create\_next\_hop(&nexthop\_id, switch\_id, 3, nexthop\_attr);

//3. Register bfd session state change notification callback

void bfd\_session\_state\_change\_notification\_fn(\_In\_ uint32\_t count,\_In\_ const sai\_bfd\_session\_state\_notification\_t \*data)

{

uint32\_t i = 0;

for(i = 0;i<count;i++)

{

//process every bfd session when state machine change

}

return;

}

sai\_api\_query(SAI\_API\_SWITCH, switch\_api);

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

switch\_attr[0].id = SAI\_SWITCH\_ATTR\_BFD\_SESSION\_STATE\_CHANGE\_NOTIFY;

switch\_attr[0].value.ptr = bfd\_session\_state\_change\_notification\_fn;

switch\_api->set\_switch\_attribute(switch\_id, &switch\_attr);

//4. Create multihop IP BFD session

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.booldata = true;

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\_IPHDR\_VERSION;

bfd\_attr[4].value.u8 = ipv4;

bfd\_attr[5].id = SAI\_BFD\_SESSION\_ATTR\_MULTIPLIER;

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

bfd\_attr[6].id =SAI\_BFD\_SESSION\_ATTR\_SRC\_IP\_ADDRESS;

bfd\_attr[6].value.ipaddr.addr.ip4 = 0x01010101;

bfd\_attr[6].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

bfd\_attr[7].id =SAI\_BFD\_SESSION\_ATTR\_DST\_IP\_ADDRESS;

bfd\_attr[7].value.ipaddr.addr.ip4 = 0x01010102;

bfd\_attr[7].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

bfd\_attr[8].id = SAI\_BFD\_SESSION\_ATTR\_MIN\_TX;

bfd\_attr[8].value.s32 = 500;

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

bfd\_attr[9].value.s32 = 500;

bfd\_attr[10].id = SAI\_BFD\_SESSION\_ATTR\_MULTIHOP;

bfd\_attr[10].value.booldata = true;

bfd\_attr[10].id = SAI\_BFD\_SESSION\_ATTR\_NEXT\_HOP\_ID;

bfd\_attr[10].value.oid = nexthop\_id;

bfd\_api->create\_bfd\_session(&bfd\_session, 11, bfd\_attr);

//5. Enable send BFD packet

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

bfd\_attr[0].value.booldata = true;

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

### R2 configuration

//1. Create rif on port1

sai\_api\_query(SAI\_API\_ROUTER\_INTERFACE, &rif\_api);

sai\_object\_id\_t rif\_id = 0;

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

rif\_attr[0].id = SAI\_ROUTER\_INTERFACE\_ATTR\_TYPE;

rif\_attr[0].value.s32 = SAI\_ROUTER\_INTERFACE\_TYPE\_PORT;

rif\_attr[1].id = SAI\_ROUTER\_INTERFACE\_ATTR\_PORT\_ID;

rif\_attr[1].value.oid = port1;

rif\_api->create\_router\_interface(&rif\_id, switch\_id, 2, rif\_attr);

//2. Create next hop for bfd outgoing, and ip address is R1

sai\_object\_id\_t nexthop\_id = 0;

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

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

nexthop\_attr[0].value.s32 = SAI\_NEXT\_HOP\_TYPE\_IP;

nexthop\_attr[1].id = SAI\_NEXT\_HOP\_ATTR\_IP;

nexthop\_attr[1].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

nexthop\_attr[1].value.ipaddr.ip4 = 0x01010101;

nexthop\_attr[2].id = SAI\_NEXT\_HOP\_ATTR\_ROUTER\_INTERFACE\_ID;

nexthop\_attr[2].value.oid = rif\_id;

nexthop\_api->create\_next\_hop(&nexthop\_id, switch\_id, 3, nexthop\_attr);

//3. Register bfd session state change notification callback

void bfd\_session\_state\_change\_notification\_fn(\_In\_ uint32\_t count,\_In\_ const sai\_bfd\_session\_state\_notification\_t \*data)

{

uint32\_t i = 0;

for(i = 0;i<count;i++)

{

//process every bfd session

}

return;

}

sai\_api\_query(SAI\_API\_SWITCH, switch\_api);

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

switch\_attr[0].id = SAI\_SWITCH\_ATTR\_BFD\_SESSION\_STATE\_CHANGE\_NOTIFY;

switch\_attr[0].value.ptr = bfd\_session\_state\_change\_notification\_fn;

switch\_api->set\_switch\_attribute(switch\_id, &switch\_attr);

//4. Create multihop IP BFD session

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.booldata = true;

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

bfd\_attr[2].value.s32 = 9345;

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

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

bfd\_attr[4].id = SAI\_BFD\_SESSION\_ATTR\_IPHDR\_VERSION;

bfd\_attr[4].value.u8 = ipv4;

bfd\_attr[5].id = SAI\_BFD\_SESSION\_ATTR\_MULTIPLIER;

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

bfd\_attr[6].id =SAI\_BFD\_SESSION\_ATTR\_SRC\_IP\_ADDRESS;

bfd\_attr[6].value.ipaddr.addr.ip4 = 0x01010102;

bfd\_attr[6].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

bfd\_attr[7].id =SAI\_BFD\_SESSION\_ATTR\_DST\_IP\_ADDRESS;

bfd\_attr[7].value.ipaddr.addr.ip4 = 0x01010101;

bfd\_attr[7].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

bfd\_attr[8].id = SAI\_BFD\_SESSION\_ATTR\_MIN\_TX;

bfd\_attr[8].value.s32 = 500;

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

bfd\_attr[9].value.s32 = 500;

bfd\_attr[10].id = SAI\_BFD\_SESSION\_ATTR\_MULTIHOP;

bfd\_attr[10].value.booldata = true;

bfd\_attr[10].id = SAI\_BFD\_SESSION\_ATTR\_NEXT\_HOP\_ID;

bfd\_attr[10].value.oid = nexthop\_id;

bfd\_api->create\_bfd\_session(&bfd\_session, 11, bfd\_attr);

//5. Enable send BFD packet

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

bfd\_attr[0].value.booldata = true;

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

### Remote discriminator learning

Some timers, local device do not know remote MEP’s discriminator, then need learning remote discriminator by the receiving packet, as shown in the figure:



1. At first, Create BFD session with SAI\_BFD\_SESSION\_ATTR\_REMOTE\_DISCRIMINATOR value 0

Such as R1:

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

bfd\_attr[0].value.s32 = 2893;

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

bfd\_attr[0].value.s32 =0;

R2:

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

bfd\_attr[0].value.s32 = 9345;

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

bfd\_attr[0].value.s32 =0;

1. create a BFD session with SAI\_BFD\_SESSION\_ATTR\_LOCAL\_DISCRIMINATOR and SAI\_BFD\_SESSION\_ATTR\_REMOTE\_DISCRIMINATOR value 0. This session use for identify the BFD packet with your discriminator = 0, and send the packet to CPU with reason SAI\_HOSTIF\_TRAP\_TYPE\_LEARNING\_BFD. When adapter receive the BFD packet, parser my discriminator and set it to corresponding BFD session.

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

bfd\_attr[0].value.s32 = 0;

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

bfd\_attr[0].value.s32 =0;

## Remove BFD Session

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

# Appendix (review suggestions)

## Review V0.2

### Suggestion by Centec

1. The saibfd as the core of BFD,, it only needs to provide APIs for configuring BFD parameters, including IP header, BFD Payload, and desired rx/tx interval,and so on.

2.BFD uses Nexthop for packet encapsulation and shares the same nexthop with data packets to achieve the same editing behavior of the outer packet header,It is also easier to extend to support other types of BFD such as MPLS BFD / VCCV BFD / VXLAN BFD , Micro BFD, S-BFD and so on.

3. Detail :

Suggestion 1: Remove sai\_bfd\_encapsulation\_type\_t, and add SAI\_BFD\_SESSION\_ATTR\_NEXT\_HOP\_ID . /\*\*

\* @brief For BFD transmit, include destination port and Encapsulation format

\*

\* @type sai\_object\_id\_t

\* @flags CREATE\_AND\_SET

\*/SAI\_BFD\_SESSION\_ATTR\_NEXT\_HOP\_ID

Suggestion 2: Remove the following attribute due to 1

SAI\_BFD\_SESSION\_ATTR\_VLAN\_TPID

SAI\_BFD\_SESSION\_ATTR\_VLAN\_ID

SAI\_BFD\_SESSION\_ATTR\_VLAN\_PRI

SAI\_BFD\_SESSION\_ATTR\_VLAN\_CFI

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

SAI\_BFD\_SESSION\_ATTR\_TUNNEL\_TOS

SAI\_BFD\_SESSION\_ATTR\_TUNNEL\_TTL

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

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

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

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

Suggestion 3: Remove SAI\_BFD\_SESSION\_ATTR\_REMOTE\_MIN\_TX and

SAI\_BFD\_SESSION\_ATTR\_REMOTE\_MIN\_RX, because the two attribute is remote system, the local no need to care.

Suggestion 4: Add two actual rx/tx interval of local .

/\*\*

\* @brief Actual Transmit interval in microseconds

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

+ SAI\_BFD\_SESSION\_ATTR\_ACTUAL\_TX,

/\*\*

\* @brief Actual Receive interval in microseconds

\*

\* @type sai\_uint32\_t

\* @flags READ\_ONLY

\*/

+ SAI\_BFD\_SESSION\_ATTR\_ACTUAL\_RX,

/\*\*

\* @brief Enable transmit BFD packet periodic in Asynchronous mode

\*

\* @type bool

\* @flags CREATE\_AND\_SET

\* @default false

\*/

+ SAI\_BFD\_SESSION\_ATTR\_TX\_ENABLE,

### Answer by DELL

1. Suggestion 1 & 2 – I agree with them
2. Suggestion 3 – It is a read only attribute, which might be required for show purposes at least.

[Centec] OK

1. Suggestion 4 – Regarding the Actual TX/RX interval, what is your expectation. Do you expect HW to negotiate the actual intervals or the BFD SW application will negotiate the intervals and program the HW. In proposal SAI\_BFD\_SESSION\_ATTR\_MIN\_TX/RX is the actual intervals

[Centec] Yes, I expect HW to negotiate the actual intervals or the BFD SW application (vendor's SDK or SAI) will negotiate the intervals. My previous understanding, the SAI\_BFD\_SESSION\_ATTR\_REMOTE\_MIN\_TX/SAI\_BFD\_SESSION\_ATTR\_REMOTE\_MIN\_RX are remore system desired Min TX Interval and Required Min RX Interval; and SAI\_BFD\_SESSION\_ATTR\_MIN\_TX/SAI\_BFD\_SESSION\_ATTR\_MIN\_RX should be used for local system desired Min TX Interval and Required Min RX Interval. So I will revise its explanation in saibfd.h.