

|  |
| --- |
|  |

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **Inband Network Telemetry** |
| **Authors** | **Barefoot Networks** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **01/18/2016** |
| **SAI-Version** | **V0.9.2** |

1 Overview 3

1.1 INT Pipeline Model 3

1.2 INT Session 4

1.3 TELEMETRY-ACE (Access Control Entry) 4

1.4 INT Source 4

1.5 INT Transit 5

1.6 INT Sink 5

2 Specification 5

2.1 saiint.h (new file) 5

2.2 Changes to saitypes.h 7

2.3 Changes to saiacl.h 8

3 Examples 8

3.1 Create INT Source Session 8

3.2 Create INT Sink Session 9

3.3 Create INT Transit Session 9

3.4 Enable INT Transit on all traffic flows 10

3.5 Enable INT Source functionality 10

3.6 Enable INT sink functionality 11

4 References 12

# Overview

This document describes the API proposal to enable INT capabilities in a network device. A network device can be configured to operate as INT-source, INT-sink or INT-transit device. Refer to INT-specification [1] for more details. The APIs described here specify various parameters that can be used to configure and control the behavior of a network device as INT source, sink or transit.

The INT protocol does not make any assumptions about underlying network virtualization deployment, but it may use the underlay protocol headers to carry INT information. The APIs described here are independent of underlying transport used by INT.

## INT Pipeline Model

Ingress Pkt

Apply Telemetry ACLs

Add Local Telemetry Information

Strip INT Info

Add INT Headers

Pkt Processing

INT Sink

INT Source

INT Transit

INT No-op

Copy to collector (Mirroring)

INT Functionality is implemented as shown above. The INT functionality does not interfere with normal data forwarding pipeline. A Telemetry ACL entry determines the INT function - source, sink, transit that is performed on a packet. Based on this, INT information is added/removed from a packet. INT sink can make a copy of a packet so that all the INT information collected in that packet along the way can be sent to the INT collector.

## INT Session

INT session is a collection of attributes that are needed to perform INT operations. An INT session is attached to an entry in TELEMETRY-ACL to enable INT operation on a packet that matches the given entry. The INT session contains following attributes –

* Session\_type - Source, Sink, Transit. Transit type can be combined with source and sink as described later in the document. It controls addition of local telemetry information to the packet.
* Max\_hop\_count – Total number of INT-transit devices that are allowed to add their telemetry information to the packet. This is used only by INT source.
* INT\_instruction\_bitmap – A bitmap that specifies the INT instructions (i.e., the types of network telemetry information) to collect. This is used only by INT source.
* INT switch\_id – Switch id that identifies the device that is performing the INT operations. This is used when inserting switch-id attribute in the packet.

## TELEMETRY-ACE (Access Control Entry)

INT source, sink and transit functionality is enabled using a TELEMETRY-ACE.

This ACE allows the user to enable INT source, sink or transit functionality based on client flows (5-tuple), tunnels, or client and tunnel endpoints. Telemetry ACEs are created and added using existing ACL APIs. Telemetry-ACE specifies an INT action and provides associated parameters.

A match key allows the user to specify either tunnel IP addresses or client IP addresses. Use of client IP address provides a fine-grained control over the traffic being monitored compared to use of tunnel IP addresses. An INT session id attribute provided to the ACE enables appropriate INT functions on the matching packet/flow.

## INT Source

An INT source device initiates INT operation by inserting INT header into a packet and thereby instructing other INT-capable downstream network devices to add desired telemetry information into the packet.

We decouple the INT-source enablement from the network-virtualization enablement configuration of the target network. This allows for enabling the INT functionality in the network (or parts of it) even when tunnels are originated and terminated elsewhere.

Note that INT source device can add its own INT information to the packet along with initiating the INT functionality. Addition of local telemetry information is treated as INT transit function (described below). Therefore enabling INT-transit along with INT-source

Instructs the device to add local telemetry information to the packet. The information such as max\_hop\_count, INT\_instruction\_bit\_map comes from the INT session associated with the ACE.

## INT Transit

An INT transit device is a device along the path of a packet from source to sink. INT-transit device adds its own INT information to the packet as requested by an INT source. An INT header must be present in the ingress and egress packet to perform INT transit functionality. TELEMETRY-ACE with a transit INT session associated with it, is used to enable transit functionality. It is possible to use an ‘allow-all’ ACE to enable transit functionality on a device (such as spine in spine-leaf topology). The telemetry information that is added to the packet is collected from various components of the packet pipeline.

## INT Sink

An INT sink device extracts the INT information from the packet and exports it using a mirror session to either the local CPU or a remote collector. Exporting the information to the collector is controlled by mirroring session associated with ACE. This uses existing mirroring APIs and ACL for configuration.

In addition to all the INT information added by upstream devices, INT sink can provide its own telemetry information if configured to do so. Enabling INT-transit along with INT-sink on the INT\_session, enables this operation.

An INT sink device must remove all INT headers from the original packet.

As mentioned earlier, an INT endpoint (source or sink) function and the network-virtualization endpoint function need not be co-located. When INT sink device is not a terminating VTEP, it only removes INT headers from the packet. Any terminating VTEP, must remove INT information from the packet irrespective of INT sink configuration.

This ensures removal of INT information from the packet when no INT\_sink was encountered along the path. This could be a result partial/erroneous configuration or path changes in the network.

# Specification

## saiint.h (new file)

/\*\* INT session types \*/

typedef enum \_sai\_int\_session\_type\_t {

SAI\_INT\_SESSION\_TYPE\_NONE = 0,

SAI\_INT\_SESSION\_TYPE\_SOURCE,

SAI\_INT\_SESSION\_TYPE\_TRANSIT,

SAI\_INT\_SESSION\_TYPE\_SINK,

} sai\_int\_session\_type\_t;

/\*\* INT instructions defined in INT specification. Each bit represents the information that is collected \*/

typedef enum \_sai\_int\_instruction\_t {

SAI\_INT\_INST\_SWITCH\_ID = 0,

SAI\_INT\_INST\_INGRESS\_PORT\_ID,

SAI\_INT\_INST\_HOP\_LATENCY,

SAI\_INT\_INST\_QUEUE\_OCCUPANCY,

SAI\_INT\_INST\_INGRESS\_TIMESTAMP,

SAI\_INT\_INST\_EGRESS\_PORT\_ID,

SAI\_INT\_INST\_QUEUE\_CONGESTION,

SAI\_INT\_INST\_TX\_UTILIZATION,

} sai\_int\_instruction\_t;

typedef enum \_sai\_int\_session\_attr\_t

/\*\* session\_type [sai\_u32\_list\_t(sai\_int\_session\_type)] (CREATE\_AND\_SET) (default to an empty list) \*/

SAI\_INT\_SESSION\_ATTR\_TYPE\_LIST,

/\*\* Mandatory for INT source \*/

SAI\_INT\_SESSION\_ATTR\_MAX\_HOP\_COUNT,

/\* Mandatory for INT source [sai\_u32\_list\_t(sai\_int\_instruction\_t)] \*/

SAI\_INT\_SESSION\_ATTR\_INT\_INST\_LIST,

/\*\* Mandatory for INT transit \*/

SAI\_INT\_SESSION\_ATTR\_INT\_SWITCH\_ID,

/\*\* optional – default = 0 \*/

SAI\_INT\_REPLICATION,

/\*\* optional – default = 0 \*/

SAI\_INT\_VERSION,

} sai\_int\_session\_attr\_t;

/\*\*

\* @brief Create INT session.

\*

\* @param[out] session\_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\_int\_session\_fn)(

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

\_In\_ uint32\_t attr\_count,

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

/\*\*

\* @brief Remove INT session.

\*

\* @param[in] session\_id

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

\* error code is returned.

\*/

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

\_In\_ sai\_object\_id\_t session\_id);

/\*\*

\* @brief Set INT session attributes.

\*

\* @param[in] session\_id INT 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\_int\_session\_attribute\_fn)(

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

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

/\*\*

\* @brief Get INT session attributes.

\*

\* @param[in] 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\_int\_session\_attribute\_fn)(

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

\_In\_ uint32\_t attr\_count,

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

typedef struct \_sai\_int\_api\_t

{

sai\_create\_int\_session\_fn create\_int\_session;

sai\_remove\_int\_session\_fn remove\_int\_session;

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

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

} sai\_int\_api\_t;

## Changes to saitypes.h

/\*\*

\* @brief Defines a INT session type list datastructure

\*/

typedef struct \_sai\_int\_session\_type\_list\_t {

/\*\* Number of types\*/

uint32\_t type\_count;

/\*\* List of types\*/

sai\_int\_session\_type\_t \*type\_list;

} sai\_int\_session\_type\_list\_t;

/\*\*

\* @brief Defines a INT session instruction list datastructure

\*/

typedef struct \_sai\_int\_session\_inst\_list\_t {

/\*\* Number of instructions \*/

uint32\_t inst\_count;

/\*\* List of instructions \*/

sai\_int\_session\_inst\_t \*inst\_list;

} sai\_int\_session\_inst\_list\_t;

/\*\*

\* @brief Data Type to use enum's as attribute value is sai\_int32\_t s32

\*

\*/

typedef union {

sai\_int\_session\_type\_list\_t int\_session\_type\_list;

sai\_int\_session\_inst\_list\_t int\_session\_inst\_list;

} sai\_attribute\_value\_t;

## Changes to saiacl.h

typedef enum \_sai\_acl\_entry\_attr\_t

{

/\*\* Enable In-band Telemetry Action - Transit \*/

SAI\_ACL\_ENRTY\_ATTR\_ACTION\_INT\_TRANSIT\_ENABLE,

/\*\* Enable In-band Telemetry Action - Source \*/

SAI\_ACL\_ENRTY\_ATTR\_ACTION\_INT\_SRC\_ENABLE,

/\*\* Enable In-band Telemetry Action - Sink \*/

SAI\_ACL\_ENRTY\_ATTR\_ACTION\_INT\_SINK\_ENABLE,

} sai\_acl\_entry\_attr\_t;

# Examples

## Create INT Source Session

/\*\* SAI create INT source session

\* enable transit - add local telemetry info

\* hop\_count = 8

\* instruction mask = switch\_id, hop\_latency, queue\_occupancy

\*/

sai\_api\_query(SAI\_API\_INT, &sai\_int\_api);

sai\_attribute\_t attr[4] = {0, };

sai\_object\_id\_t session\_id;

sai\_int\_session\_type\_t type\_list[2] = {0, };

sai\_int\_session\_inst\_t inst\_list[3] = {0, };

type\_list[0] = SAI\_INT\_SESSION\_TYPE\_SOURCE;

type\_list[1] = SAI\_INT\_SESSION\_TYPE\_TRANSIT;

attr[0].id = SAI\_INT\_SESSION\_ATTR\_TYPE\_LIST;

attr[0].val.int\_session\_type\_list.type\_count = 2;

attr[0].val.int\_session\_type\_list.type\_list = type\_list;

attr[1].id = SAI\_INT\_SESSION\_ATTR\_MAX\_HOP\_COUNT;

attr[1].value.u32 = 8;

inst\_mask[0] = INT\_INST\_SWITCH\_ID;

inst\_mask[1] = INT\_INST\_HOP\_LATENCY;

inst\_mask[2] = INT\_INST\_QUEUE\_OCCUPANCY;

attr[2].val.int\_session\_inst\_list.type\_count = 2;

attr[2].val.int\_session\_inst\_list.inst\_list = inst\_list;

attr[2].id = SAI\_INT\_SESSION\_ATTR\_INT\_INST\_LIST;

attr[3].id = SAI\_INT\_SESSION\_ATTR\_INT\_SWITCH\_ID;

attr[3].value.u32 = switch\_id;

sai\_int\_api->sai\_create\_int\_session(&session\_id, 4, attr);

## Create INT Sink Session

/\*\* Create INT sink session, without local telemetry info enabled \*/

sai\_api\_query(SAI\_API\_INT, &sai\_int\_api);

sai\_attribute\_t attr[1] = {0, };

sai\_object\_id\_t session\_id;

sai\_int\_session\_type\_t type\_list[1] = {0, };

type\_list[0] = SAI\_INT\_SESSION\_TYPE\_SINK;

attr[0].id = SAI\_INT\_SESSION\_ATTR\_TYPE\_LIST;

attr[0].val.int\_session\_type\_list.type\_count = 1;

attr[0].val.int\_session\_type\_list.type\_list = type\_list;

sai\_int\_api->sai\_create\_int\_session(&session\_id, 1, attr);

## Create INT Transit Session

/\*\* create INT transit session \*/

sai\_api\_query(SAI\_API\_INT, &sai\_int\_api);

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

sai\_object\_id\_t session\_id;

sai\_int\_session\_type\_t type\_list[1] = {0, };

type\_list[0] = SAI\_INT\_SESSION\_TYPE\_TRANSIT;

attr[0].id = SAI\_INT\_SESSION\_ATTR\_TYPE\_LIST;

attr[0].val.int\_session\_type\_list.type\_count = 1;

attr[0].val.int\_session\_type\_list.type\_list = type\_list;

attr[1].id = SAI\_INT\_SESSION\_ATTR\_INT\_SWITCH\_ID;

attr[1].value.u32 = switch\_id;

sai\_int\_api->sai\_create\_int\_session(&session\_id, 2, attr);

## Enable INT Transit on all traffic flows

/\*\* Enable INT transit on all tunnels \*/

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

sai\_acl\_entry\_id\_t int\_acl\_rule\_id;

rule\_attr[0].id = SAI\_ACL\_ENTRY\_ATTR\_TABLE\_ID;

rule\_attr[0].value.u32 = int\_acl\_table\_id; /\* previously created \*/

rule\_attr[1].id = SAI\_ACL\_ENTRY\_ATTR\_PRIORITY;

rule\_attr[1].value.u32 = 1;

/\* Do not specify IP addresses - i.e. \* (or masked) \*/

/\* Action data \*/

rule\_attr[2].id = SAI\_ACL\_ENRTY\_ATTR\_ACTION\_INT\_TRANSIT\_ENABLE;

rule\_attr[2].value.aclaction.enable = true;

rule\_attr[2].value.aclaction.parameter.u32 = int\_transit\_session\_id;

sai\_acl\_api->create\_acl\_rule (&int\_acl\_rule\_id, 3, &rule\_attr);

## Enable INT Source functionality

/\*\* Enable INT source on a specific client flow specified by -

\* {sip, dip, vrf|bd, proto, src\_port, dst\_port}

\*/

sai\_attribute\_t rule\_attr[11];

sai\_acl\_entry\_id\_t int\_acl\_rule\_id;

rule\_attr[0].id = SAI\_ACL\_ENTRY\_ATTR\_TABLE\_ID;

rule\_attr[0].value.u32 = int\_acl\_table\_id;

rule\_attr[1].id = SAI\_ACL\_ENTRY\_ATTR\_PRIORITY;

rule\_attr[1].value.u32 = 1;

rule\_attr[2].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_DST\_IP;

rule\_attr[2].value.aclfield.enable = true;

rule\_attr[2].value.aclfield.data.ipv4 = dip;

rule\_attr[2].value.aclfield.mask.ipv4 = (uint32\_t)-1;

rule\_attr[3].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_SRC\_IP;

rule\_attr[3].value.aclfield.enable = true;

rule\_attr[3].value.aclfield.data.ipv4 = sip;

rule\_attr[3].value.aclfield.mask.ipv4 = (uint32\_t)-1;

rule\_attr[4].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_IP\_PROTOCOL;

rule\_attr[4].value.aclfield.enable = true;

rule\_attr[4].value.aclfield.data.u8 = proto; /\* tcp, udp, ... \*/

rule\_attr[4].value.aclfield.mask.u8 = (uint8\_t)-1;

rule\_attr[5].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_L4\_DST\_PORT;

rule\_attr[5].value.aclfield.enable = true;

rule\_attr[5].value.aclfield.data.u8 = dst\_port;

rule\_attr[5].value.aclfield.mask.u8 = (uint8\_t)-1;

rule\_attr[6].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_L4\_SRC\_PORT;

rule\_attr[6].value.aclfield.enable = true;

rule\_attr[6].value.aclfield.data.u8 = src\_port;

rule\_attr[6].value.aclfield.mask.u8 = (uint8\_t)-1;

/\* Action data \*/

rule\_attr[7].id = SAI\_ACL\_ENRTY\_ATTR\_ACTION\_INT\_SRC\_ENABLE;

rule\_attr[7].value.aclaction.enable = true;

rule\_attr[7].value.parameter.u32 = int\_src\_session\_id; /\* previously created \*/

sai\_acl\_api->create\_acl\_rule (&int\_acl\_rule\_id, 8, &rule\_attr);

## Enable INT sink functionality

/\*\* Enable INT sink on a specific client flow specified by -

\* {sip, dip, vrf|bd, proto, src\_port, dst\_port}

\* with mirror\_id to send data to collector

\*/

sai\_attribute\_t rule\_attr[9];

sai\_acl\_entry\_id\_t int\_acl\_rule\_id;

rule\_attr[0].id = SAI\_ACL\_ENTRY\_ATTR\_TABLE\_ID;

rule\_attr[0].value.u32 = int\_acl\_table\_id;

rule\_attr[1].id = SAI\_ACL\_ENTRY\_ATTR\_PRIORITY;

rule\_attr[1].value.u32 = 1;

rule\_attr[2].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_DST\_IP;

rule\_attr[2].value.aclfield.enable = true;

rule\_attr[2].value.aclfield.data.ipv4 = dip;

rule\_attr[2].value.aclfield.mask.ipv4 = (uint32\_t)-1;

rule\_attr[3].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_SRC\_IP;

rule\_attr[3].value.aclfield.enable = true;

rule\_attr[3].value.aclfield.data.ipv4 = sip;

rule\_attr[3].value.aclfield.mask.ipv4 = (uint32\_t)-1;

rule\_attr[4].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_IP\_PROTOCOL;

rule\_attr[4].value.aclfield.enable = true;

rule\_attr[4].value.aclfield.data.u8 = proto; /\* tcp, udp, ... \*/

rule\_attr[4].value.aclfield.mask.u8 = (uint8\_t)-1;

rule\_attr[5].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_L4\_DST\_PORT;

rule\_attr[5].value.aclfield.enable = true;

rule\_attr[5].value.aclfield.data.u8 = dst\_port;

rule\_attr[5].value.aclfield.mask.u8 = (uint8\_t)-1;

rule\_attr[6].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_L4\_SRC\_PORT;

rule\_attr[6].value.aclfield.enable = true;

rule\_attr[6].value.aclfield.data.u8 = src\_port;

rule\_attr[6].value.aclfield.mask.u8 = (uint8\_t)-1;

/\*\* Action data \*/

rule\_attr[7].id = SAI\_ACL\_ENRTY\_ATTR\_ACTION\_INT\_SINK\_ENABLE;

rule\_attr[7].value.aclaction.enable = true;

rule\_attr[7].value.parameter.u32 = int\_sink\_session\_id; /\* previously created \*/

/\*\* Mirror to collector \*/

rule\_attr[8].id = SAI\_ACL\_ENRTY\_ATTR\_ACTION\_MIRROR\_INGRESS;

rule\_attr[8].value.aclaction.enable = true;

rule\_attr[8].value.parameter.u32 = int\_mirror\_session\_id; /\* previously created \*/

sai\_acl\_api->create\_acl\_rule (&int\_acl\_rule\_id, 9, &rule\_attr);

# References

1. INT specification – http://p4.org/wp-content/uploads/fixed/INT/INT-current-spec.pdf