

Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **SAI Telemetry and Monitoring (TAM)** |
| **Authors** | **Broadcom Limited** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **10 July 2018** |
| **SAI-Version** | **1.3** |

Contents

[List of Changes 3](#_Toc519096386)

[1 Overview 5](#_Toc519096387)

[2 Proposal 5](#_Toc519096388)

[2.1 TAM API abstraction 5](#_Toc519096389)

[2.1.1 Data Push model 7](#_Toc519096390)

[2.1.2 Data Pull model 7](#_Toc519096391)

[2.2 Changes from TAM 1.0 spec 8](#_Toc519096392)

[3 Specification 9](#_Toc519096393)

[3.1 SAI API and TAM Objects 9](#_Toc519096394)

[4 Examples 11](#_Toc519096395)

[4.1 Creating a Synthetic Probe session 11](#_Toc519096396)

[4.2 Creating a flow telemetry session 14](#_Toc519096397)

[4.3 Creating an event object 18](#_Toc519096398)

[4.4 Creating a microburst detection object and sending the histogram report to localhost 22](#_Toc519096399)

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 1 | Proposal for SAI Telemetry and Monitoring 1.0 | Broadcom Limited | 20 Dec 2016 |
| 2 | Updated to incorporate feedback comments | Broadcom Limited | 25 Jan 2017 |
| 3 | Proposal for SAI Telemetry and Monitoring 2.0 | Broadcom Limited | 21 Mar 2018 |
| 4 | Updated to incorporate feedback comments | Broadcom Limited | 10 July 2018 |

License

© 2014 Microsoft Corporation, Dell Inc., Facebook, Inc, Broadcom Limited, 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

This spec enhances the existing TAM (Telemetry and Monitoring) spec to address overall telemetry, events, and reporting use cases.

Scope of this spec is data plane objects. Control plane objects need more involved data set and data decoration method and are/may be streamed on administrative channels like NetCONF, SNMP, gRPC etc.

# Proposal

Proposed enhancement to Telemetry and Monitoring (TAM) is a hierarchical API model designed with following goals:

* Express top level Telemetry and Monitoring domain
* Remain backward compatible for application with minimal or no change in the application code
* Provide extensibility for new functions/capability in each domain/sub domain and always remain backward compatible
* Remove any hardcoded assumptions about domain or domain’s capabilities
* Provide full abstraction for operators to dynamically learn the number of domains supported within a networking element
* Provide full abstraction for operators to dynamically learn domains capabilities
* Provide flexibility to add new APIs for a given domain/sub domain
* Support local mathematical functions for hierarchical analysis

This document doesn’t cover control plane telemetry, although the API itself is not constrained and can easily be extended.

## TAM API abstraction

APIs are used to subscribe for a given class. Each class has various data objects with various data attributes. Subscription is done at the data object level to keep things simple. Subscription at data attribute granularity will severely overload the system in terms of processing and keeping subscribed state.

Once an API subscribes for a given data object for a given system, all data attributes supported by the system will be reported.

Capability API MUST be supported at system level and/or data object level so that controller or NOS can create an indexing scheme prior to collecting the data and learning the attributes dynamically for mixed vendor deployments.

Ability to dynamically learn the data objects will provide flexibility to both producer and collector subsystems to scale dynamically.

Ability to dynamically learn the data objects will also help in mixed vendor deployments where different systems can support different set of data objects.

Granular subscription for a data set within a domain is also supported. The controller can subscribe for per port per queue or per port ingress or egress stats.

Control Flow for a controller or collector subsystem looks like this.

Invoke Capability API

Gather Data Obj/Attr

Create DB Index

Start Collecting Data

## Data Push model

Here is an abstract diagram of PUSH model of the API and switch function



Figure: PUSH model of Telemetry API

Once data is subscribed, data plane can publish the data to external collector or to a local collector hosted within the NOS.

Additionally data plane can perform mathematical functions like histogram, mean, mode or summary to present a concise view of a data set.

Data subscription is done at a fine granularity so as not to overwhelm the collector specially if it is a local function in NOS.

## Data Pull model

TBD: What are the APIs for retrieving the data from the chip.

## Data model

TBD: What is the data model for streaming and retrieval of the data.

## Granular Subscription of Data

TBD: How to stream or retrieve a granular data type for eg only ingress port stats or ingress error stats.

## New Telemetry Type Creation Work Flow

TBD: How to add a new data type in the existing API

## Changes from TAM 1.0 spec

SAI\_OBJECT\_TYPE\_TAM                      = 60,

~~SAI\_OBJECT\_TYPE\_TAM\_STAT                 = 61~~, <- folded in TAM\_TELEMETRY

~~SAI\_OBJECT\_TYPE\_TAM\_SNAPSHOT             = 62~~, <- folded in TAM\_EVENT

~~SAI\_OBJECT\_TYPE\_TAM\_TRANSPORTER          = 63~~, <- folded in TAM\_TRANSPORT

~~SAI\_OBJECT\_TYPE\_TAM\_THRESHOLD            = 64~~, <- folded in TAM\_EVENT

~~SAI\_OBJECT\_TYPE\_TAM\_HISTOGRAM            = 68~~, <- folded in TAM\_REPORT (it’s a type of report)

~~SAI\_OBJECT\_TYPE\_TAM\_MICROBURST           = 69~~, <- folded in TAM\_EVENT

SAI\_OBJECT\_TYPE\_TAM\_MATH\_FUNC            = 76,

SAI\_OBJECT\_TYPE\_TAM\_REPORT               = 77,

SAI\_OBJECT\_TYPE\_TAM\_EVENT\_THRESHOLD = 78,

SAI\_OBJECT\_TYPE\_TAM\_TEL\_TYPE             = 79,

SAI\_OBJECT\_TYPE\_TAM\_TRANSPORT            = 80,

SAI\_OBJECT\_TYPE\_TAM\_IFA\_HDR              = 81,

SAI\_OBJECT\_TYPE\_TAM\_IFA\_PROBE            = 82,

SAI\_OBJECT\_TYPE\_TAM\_PROBE                = 83,

SAI\_OBJECT\_TYPE\_TAM\_TELEMETRY            = 84,

SAI\_OBJECT\_TYPE\_TAM\_COLLECTOR            = 85,

SAI\_OBJECT\_TYPE\_TAM\_EVENT\_ACTION         = 86,

SAI\_OBJECT\_TYPE\_TAM\_EVENT                = 87,

# Specification

## SAI API and TAM Objects

Following object types and attributes are defined

* Capability objects
  + Telemetry Capability
  + Event Capability
  + Probe Capability
  + Data Report Capability
  + Reporting Rate Capability
* Telemetry object
  + Telemetry Type
    - INT telemetry
    - Flow telemetry
      * Flow id
      * Classification enable
      * Duration
      * Size
      * Action
    - Switch telemetry
      * Enable port stats
      * Enable queue stats
      * Enable VOQ stats
      * Enable MMU stats
      * Enable Fabric stats
      * Enable filter stats
      * Enable Resource utilization stats
    - Fabric telemetry
      * Collect fabric queue stats
    - NetworkingElement (NE) telemetry
    - Math Func API
  + Collector List
  + Report id
  + Reporting unit
  + Reporting interval
* Probe object
  + Probe Type
    - Synthetic Probe
    - Sampled Probe
  + Probe Header object
    - Markers
    - Version
    - Replication bits
    - Copy, Hop count exceed, control bits
    - Message type
    - Flag
    - Request and action vectors
    - Hop limit
    - Max header length
    - Current header length
    - Senders handle
    - Sequence number
  + IFA Probe
    - IP header verson
    - TOS
    - TTL
    - SIP
    - DIP
    - Protocol type
    - Src, dst port
    - UDP header length
    - Header object
* Collector object
  + Transport Type
    - TCP
    - UDP
    - INT
    - gRPC
  + Transport config
    - Localhost
    - DIP
    - SIP
    - RouterID
    - DSCP
    - TruncateSize
* Event object
  + Event Report
  + Event Action List
    - QoS Q assignment
    - Nexthop
    - Report
* Data Report object
  + Report Type
    - SFLOW
    - IPFIX
    - ProtoBuf
    - TRHIFT
    - INT
    - HISTOGRAM
    - VendorEXTN
  + Histogram number of bins
  + Histogram bin boundary
* Threshold object
  + High water mark
  + Low water mark
  + Latency
  + Rate
  + Absolute Value
* Transport object
  + Transport Type
    - TCP
    - UDP
    - INT
    - gRPC
  + Src, dst port
  + Transport Authentication
    - SSL
    - TLS

# Examples

/\* Object ids for TAM object types \*/

sai\_object\_id\_t sai\_tam\_object;

sai\_object\_id\_t sai\_tam\_probe\_obj;

sai\_object\_id\_t sai\_tam\_math\_func\_obj;

sai\_object\_id\_t sai\_tam\_transport\_obj,

sai\_object\_id\_t sai\_tam\_threshold\_obj;

sai\_object\_id\_t sai\_tam\_collector\_obj;

sai\_object\_id\_t sai\_tam\_flow\_tel\_type\_obj;

sai\_object\_id\_t sai\_tam\_telemetry\_obj;

sai\_object\_id\_t sai\_ifa\_hdr\_obj;

sai\_object\_id\_t sai\_ifa\_probe\_obj;

sai\_object\_id\_t sai\_tam\_report\_obj;

sai\_object\_id\_t sai\_tam\_event\_action\_obj;

sai\_object\_id\_t sai\_tam\_event\_obj;

sai\_attribute\_t sai\_attr\_list[20];

/\* reference to the switch that was created elsewhere \*/

extern sai\_object\_id\_t  switch\_id;

## Creating a Synthetic Probe session

*/\* Step 1: Create an IFA header:*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_IFA\_HDR\_ATTR\_MARKER\_1;

sai\_attr\_list[0].value.u32 = 0x10cc10cc;

sai\_attr\_list[1].id = SAI\_TAM\_IFA\_HDR\_ATTR\_MARKER\_2;

sai\_attr\_list[1].value.u32 = 0xdead10cc;

sai\_attr\_list[2].id = SAI\_TAM\_IFA\_HDR\_ATTR\_VER;

sai\_attr\_list[2].value.u16 = 1;

sai\_attr\_list[3].id = SAI\_TAM\_IFA\_HDR\_ATTR\_TAM\_ID;

sai\_attr\_list[3].value.oid = sai\_tam\_object;

attr\_count = 4;

sai\_create\_tam\_ifa\_hdr\_fn(

&sai\_ifa\_hdr\_obj,

switch\_id,

attr\_Count,

sai\_attr\_list);

*/\* Step 2: Create a IFA probe:*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_IFA\_PROBE\_ATTR\_HDR\_IPHDR\_VERSION;

sai\_attr\_list[0].value.u8 = 4;

sai\_attr\_list[1].id = SAI\_TAM\_IFA\_PROBE\_ATTR\_SRC\_IP\_ADDRESS;

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

sai\_attr\_list[1].value.ipaddr.ip4 = 0x0101010a;

sai\_attr\_list[2].id = SAI\_TAM\_IFA\_PROBE\_ATTR\_DST\_IP\_ADDRESS;

sai\_attr\_list[2].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

sai\_attr\_list[2].value.ipaddr.ip4 = 0x0101010b;

sai\_attr\_list[3].id = SAI\_TAM\_IFA\_PROBE\_ATTR\_PROTOCOL\_TYPE;

sai\_attr\_list[3].value.u16 = 17; /\* UDP \*/

sai\_attr\_list[4].id = SAI\_TAM\_IFA\_PROBE\_ATTR\_PROTO\_SPORT;

sai\_attr\_list[4].value.u16 = 0xB1FA;

sai\_attr\_list[5].id = SAI\_TAM\_IFA\_PROBE\_ATTR\_PROTO\_DPORT;

sai\_attr\_list[5].value.u16 = 0xC1FA;

sai\_attr\_list[6].id = SAI\_TAM\_IFA\_PROBE\_ATTR\_LEN;

sai\_attr\_list[6].value.u16 = 128;

sai\_attr\_list[7].id = SAI\_TAM\_IFA\_PROBE\_ATTR\_IFA\_HDR;

sai\_attr\_list[7].value.oid = sai\_ifa\_hdr\_obj;

sai\_attr\_list[8].id = SAI\_TAM\_IFA\_PROBE\_ATTR\_TAM\_ID;

sai\_attr\_list[8].value.oid = sai\_tam\_object;

attr\_count = 9;

sai\_create\_tam\_ifa\_probe\_fn(

&sai\_ifa\_probe\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 3: Create synthetic probe object:*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_PROBE\_ATTR\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_PROBE\_TYPE\_SYNTHETIC;

sai\_attr\_list[1].id = SAI\_TAM\_PROBE\_ATTR\_HDR;

sai\_attr\_list[1].value.oid = sai\_ifa\_probe\_obj;

sai\_attr\_list[2].id = SAI\_TAM\_PROBE\_ATTR\_HDR\_TAM\_ID;

sai\_attr\_list[2].value.oid = sai\_tam\_object;

attr\_count = 3;

sai\_create\_tam\_probe\_fn(

&sai\_tam\_probe\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

## Creating a flow telemetry session

*/\* Step 1: Create a transport object:*

*\* ---------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_TRANSPORT\_ATTR\_TRANSPORT\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_TRANSPORT\_TYPE\_UDP;

sai\_attr\_list[1].id = SAI\_TAM\_TRANSPORT\_ATTR\_TAM\_ID;

sai\_attr\_list[1].value.oid = sai\_tam\_object;

sai\_create\_tam\_transport\_fn(&sai\_tam\_transport\_obj, switch\_id, 2, sai\_attr\_list);

*/\* Step 2: Create a collector object:*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_COLLECTOR\_ATTR\_SRC\_IP;

sai\_attr\_list[0].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

sai\_attr\_list[0].value.ipaddr.ip4 = 0x0101010a;

sai\_attr\_list[1].id = SAI\_TAM\_COLLECTOR\_ATTR\_DST\_IP;

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

sai\_attr\_list[1].value.ipaddr.ip4 = 0x0101010b;

sai\_attr\_list[2].id = SAI\_TAM\_TRANSPORT\_ATTR;

sai\_attr\_list[2].value.oid = sai\_tam\_transport\_obj;

sai\_attr\_list[3].id = SAI\_TAM\_COLLECTOR\_ATTR\_DSCP\_VALUE;

sai\_attr\_list[3].value.u8 = 16;

sai\_attr\_list[4].id = SAI\_TAM\_COLLECTOR\_ATTR\_TAM\_ID;

sai\_attr\_list[4].value.oid = sai\_tam\_object;

attr\_count = 5;

sai\_create\_tam\_collector\_fn(

&sai\_tam\_collector\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 3: Create a math function*

*\* ---------------------------------------- \*/*

/\* create math function for rate computation \*/

sai\_attr\_list[0].id = SAI\_TAM\_MATH\_FUNC\_ATTR\_TAM\_TEL\_MATH\_FUNC\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_TEL\_MATH\_FUNC\_TYPE\_RATE;

sai\_attr\_list[1].id = SAI\_TAM\_MATH\_FUNC\_ATTR\_TAM\_ID;

sai\_attr\_list[1].value.oid = sai\_tam\_object;

attr\_count = 2;

sai\_create\_tam\_math\_func\_fn(

&sai\_tam\_math\_func\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 4: Create a flow telemetry type object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_TEL\_TYPE\_ATTR\_TAM\_TELEMETRY\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_TELEMETRY\_TYPE\_FLOW;

sai\_attr\_list[1].id = SAI\_TAM\_TEL\_TYPE\_ATTR\_FLOW\_ID;

sai\_attr\_list[1].value.u32 = 0x12345678;

sai\_attr\_list[2].id = SAI\_TAM\_TEL\_TYPE\_ATTR\_MATH\_FUNC;

sai\_attr\_list[2].value.oid = sai\_tam\_math\_func\_obj;

sai\_attr\_list[3].id = SAI\_TAM\_TEL\_TYPE\_ATTR\_TAM\_ID;

sai\_attr\_list[3].value.oid = sai\_tam\_object;

attr\_count = 4;

sai\_create\_tam\_tel\_type\_fn(

&sai\_tam\_flow\_tel\_type\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 5: Create report object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_REPORT\_ATTR\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_REPORT\_TYPE\_INT;

sai\_attr\_list[1].id = SAI\_TAM\_REPORT\_ATTR\_TAM\_ID;

sai\_attr\_list[1].value.oid = sai\_tam\_object;

attr\_count = 2;

sai\_create\_tam\_report\_fn(

&sai\_tam\_report\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 6: Create telemetry object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_TELEMETRY\_ATTR\_TAM\_TYPE\_LIST;

sai\_attr\_list[0].value.objlist.count = 1;

sai\_attr\_list[0].value.objlist.list[0] = sai\_tam\_flow\_tel\_type\_obj;

sai\_attr\_list[1].id = SAI\_TAM\_TELEMETRY\_ATTR\_COLLECTOR\_LIST;

sai\_attr\_list[1].value.objlist.count = 1;

sai\_attr\_list[1].value.objlist.list[0] = sai\_tam\_collector\_obj;

sai\_attr\_list[2].id = SAI\_TAM\_TELEMETRY\_ATTR\_REPORT\_ID;

sai\_attr\_list[2].value.oid = sai\_tam\_report\_obj;

sai\_attr\_list[3].id = SAI\_TAM\_TELEMETRY\_ATTR\_TAM\_ID;

sai\_attr\_list[3].value.oid = sai\_tam\_object;

attr\_count = 4;

sai\_create\_tam\_telemetry\_fn(

&sai\_tam\_telemetry\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

## Creating an event object

*/\* Step 1: Create a transport object:*

*\* ---------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_TRANSPORT\_ATTR\_TRANSPORT\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_TRANSPORT\_TYPE\_UDP;

sai\_attr\_list[1].id = SAI\_TAM\_TRANSPORT\_ATTR\_TAM\_ID;

sai\_attr\_list[1].value.oid = sai\_tam\_object;

attr\_count = 2;

sai\_create\_tam\_transport\_fn(

&sai\_tam\_transport\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 2: Create a collector object:*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_COLLECTOR\_ATTR\_SRC\_IP;

sai\_attr\_list[0].value.ipaddr.addr\_family = SAI\_IP\_ADDR\_FAMILY\_IPV4;

sai\_attr\_list[0].value.ipaddr.ip4 = 0x0101010a;

sai\_attr\_list[1].id = SAI\_TAM\_COLLECTOR\_ATTR\_DST\_IP;

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

sai\_attr\_list[1].value.ipaddr.ip4 = 0x0101010b;

sai\_attr\_list[2].id = SAI\_TAM\_TRANSPORT\_ATTR;

sai\_attr\_list[2].value.oid = sai\_tam\_transport\_obj;

sai\_attr\_list[3].id = SAI\_TAM\_COLLECTOR\_ATTR\_DSCP\_VALUE;

sai\_attr\_list[3].value.u8 = 16;

sai\_attr\_list[4].id = SAI\_TAM\_COLLECTOR\_ATTR\_TAM\_ID;

sai\_attr\_list[4].value.oid = sai\_tam\_object;

attr\_count = 5;

sai\_create\_tam\_collector\_fn(

&sai\_tam\_collector\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 3: Create a threshold object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_EVENT\_THRESHOLD\_ATTR\_RATE;

sai\_attr\_list[0].value.u32 = 500000;

sai\_attr\_list[1].id = SAI\_TAM\_EVENT\_THRESHOLD\_ATTR\_TAM\_ID;

sai\_attr\_list[1].value.oid = sai\_tam\_object;

attr\_count = 2;

sai\_create\_tam\_event\_threshold\_fn(

&sai\_tam\_threshold\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 4: Create an event action object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_EVENT\_ACTION\_ATTR\_REPORT\_TYPE;

sai\_attr\_list[0].value.oid = sai\_tam\_report\_obj;

sai\_attr\_list[1].id = SAI\_TAM\_EVENT\_ACTION\_ATTR\_TAM\_ID;

sai\_attr\_list[1].value.oid = sai\_tam\_object;

attr\_count = 2;

sai\_create\_tam\_event\_action\_fn(

&sai\_tam\_event\_action\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Final step: stitch all to create event object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_EVENT\_ATTR\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_EVENT\_TYPE\_PACKET\_DROP;

sai\_attr\_list[1].id = SAI\_TAM\_EVENT\_ATTR\_ACTION\_LIST;

sai\_attr\_list[1].value.objlist.count = 1;

sai\_attr\_list[1].value.objlist.list[0] = sai\_tam\_event\_action\_obj;

sai\_attr\_list[2].id = SAI\_TAM\_EVENT\_ATTR\_COLLECTOR\_LIST;

sai\_attr\_list[2].value.objlist.count = 1;

sai\_attr\_list[2].value.objlist.list[0] = sai\_tam\_collector\_obj;

sai\_attr\_list[3].id = SAI\_TAM\_EVENT\_ATTR\_THRESHOLD;

sai\_attr\_list[3].value.oid = sai\_tam\_threshold\_obj;

sai\_attr\_list[4].id = SAI\_TAM\_EVENT\_ATTR\_TAM\_ID;

sai\_attr\_list[4].value.oid = sai\_tam\_object;

attr\_count = 5;

sai\_create\_tam\_event\_fn(

&sai\_tam\_event\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

## Creating a microburst detection object and sending the histogram report to localhost

*/\* Step 1: Create a transport object:*

*\* ---------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_TRANSPORT\_ATTR\_TRANSPORT\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_TRANSPORT\_TYPE\_NONE;

sai\_attr\_list[1].id = SAI\_TAM\_TRANSPORT\_ATTR\_TAM\_ID;

sai\_attr\_list[1].value.oid = sai\_tam\_object;

attr\_count = 2;

sai\_create\_tam\_transport\_fn(

&sai\_tam\_transport\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 2: Create a localhost collector object:*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_COLLECTOR\_LOCALHOST;

sai\_attr\_list[1].id = SAI\_TAM\_TRANSPORT\_ATTR;

sai\_attr\_list[1].value.oid = sai\_tam\_transport\_obj;

sai\_attr\_list[2].id = SAI\_TAM\_COLLECTOR\_ATTR\_TAM\_ID;

sai\_attr\_list[2].value.oid = sai\_tam\_object;

attr\_count = 3;

sai\_create\_tam\_collector\_fn(

&sai\_tam\_collector\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 3: Create a threshold object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_EVENT\_THRESHOLD\_ATTR\_LOW\_WATERMARK;

sai\_attr\_list[0].value.u32 = 50000;

sai\_attr\_list[1].id = SAI\_TAM\_EVENT\_THRESHOLD\_ATTR\_HIGH\_WATERMARK;

sai\_attr\_list[1].value.u32 = 450000;

sai\_attr\_list[2].id = SAI\_TAM\_EVENT\_THRESHOLD\_ATTR\_TAM\_ID;

sai\_attr\_list[2].value.oid = sai\_tam\_object;

attr\_count = 3;

sai\_create\_tam\_event\_threshold\_fn(

&sai\_tam\_threshold\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 4: Create report object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_REPORT\_ATTR\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_REPORT\_TYPE\_HISTOGRAM;

sai\_attr\_list[1].id = SAI\_TAM\_REPORT\_ATTR\_TAM\_ID;

sai\_attr\_list[1].value.oid = sai\_tam\_object;

sai\_attr\_list[2].id = SAI\_TAM\_HISTOGRAM\_ATTR\_NUMBER\_OF\_BINS;

sai\_attr\_list[2].value.u32 = 4;

sai\_attr\_list[3].id = SAI\_TAM\_HISTOGRAM\_ATTR\_BIN\_BOUNDARY;

sai\_attr\_list[3].value.list[0] = 0; /\* bin-#0 - duration from 0us to 2us \*/

sai\_attr\_list[3].value.list[1] = 2; /\* bin-#1 - duration from t1=2us to t2=5us \*/

sai\_attr\_list[3].value.list[2] = 5; /\* bin-#2 - duration from t2=5us to t3=10us \*/

sai\_attr\_list[3].value.list[3] = 10;/\* bin-#3 - duration more than t3=10us. \*/

sai\_attr\_list[3].value.list\_size = 4

attr\_count = 4;

sai\_create\_tam\_report\_fn(

&sai\_tam\_report\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Step 5: Create an event action object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_EVENT\_ACTION\_ATTR\_REPORT\_TYPE;

sai\_attr\_list[0].value.oid = sai\_tam\_report\_obj;

sai\_attr\_list[1].id = SAI\_TAM\_EVENT\_ACTION\_ATTR\_TAM\_ID;

sai\_attr\_list[1].value.oid = sai\_tam\_object;

attr\_count = 2;

sai\_create\_tam\_event\_action\_fn(

&sai\_tam\_event\_action\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);

*/\* Final step: stitch all to create microburst event object*

*\* ---------------------------------------- \*/*

sai\_attr\_list[0].id = SAI\_TAM\_EVENT\_ATTR\_TYPE;

sai\_attr\_list[0].value.s32 = SAI\_TAM\_EVENT\_TYPE\_QUEUE\_THRESHOLD;

sai\_attr\_list[1].id = SAI\_TAM\_EVENT\_ATTR\_ACTION\_LIST;

sai\_attr\_list[1].value.objlist.count = 1;

sai\_attr\_list[1].value.objlist.list[0] = sai\_tam\_event\_action\_obj;

sai\_attr\_list[2].id = SAI\_TAM\_EVENT\_ATTR\_COLLECTOR\_LIST;

sai\_attr\_list[2].value.objlist.count = 1;

sai\_attr\_list[2].value.objlist.list[0] = sai\_tam\_collector\_obj;

sai\_attr\_list[3].id = SAI\_TAM\_EVENT\_ATTR\_THRESHOLD;

sai\_attr\_list[3].value.oid = sai\_tam\_threshold\_obj;

sai\_attr\_list[4].id = SAI\_TAM\_EVENT\_ATTR\_TAM\_ID;

sai\_attr\_list[4].value.oid = sai\_tam\_object;

attr\_count = 5;

sai\_create\_tam\_event\_fn(

&sai\_tam\_event\_obj,

switch\_id,

attr\_count,

sai\_attr\_list);