# SAI Load Balancer Proposal

|  |  |
| --- | --- |
| **Title** | **SAI Load Balancer Proposal** |
| Authors | InsidePacket Inc. |
| Status | Preliminary Draft |
| Type | Standards track |
| Updated | 1/30/2020 |
| SAI-Version | 1.5 |

# Load Balancer API

This document defines a set of proposed SAI APIs to configure a Layer-4 Service Load Balancer. It follows a generic model to allow defining the individual logical components common in typical Load Balancer configurations.

## Load Balancer Deployment Use Case

A L4 Service Load Balancer (L4LB) serves as a single destination, defined by an {IP Address, TCP Port} pair, for a service running on multiple functionally equivalent endpoints. Traffic that is received on the L4LB service is distributed among multiple endpoints - servers, microservices etc. The Load Balancer IP address is often referred to as Virtual IP (VIP), and the corresponding service – as a Virtual Service (VS). The network connectivity between the Load Balancer and the end services can be direct, or indirect with other switching devices between them.

Traffic to the end services can be encapsulated using tunneling methods such as VxLAN or IP in IP.

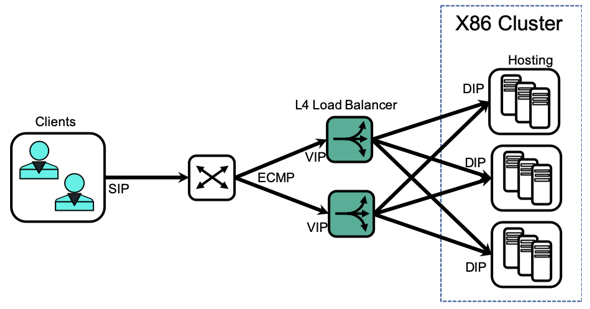
Below is a schematic representation of the Load Balancing function:

SIP

VIP

LB

DIP

A Layer-4 **Load Balancing** topology looks as follows (traffic may go through other non-balancing hops to the servers):

A variety of algorithms can be used to distribute the traffic among the servers - Uniform or Weighted Hash, Round Robin etc. A network architecture with L4 Load Balancers provides high resiliency and source-IP affinity even during changes in service endpoints availability. For example:

If an end-server is removed, the L4LB will redirect new connections to remaining servers without impacting the existing traffic while preserving connection affinity allowing active connections to expire:

Existing Connections

Server Shutdown

New connections

Stage 1 – Load Balancing

Stage 2 – Mark Server for Shutdown

Existing Connections

### Service State Transitions

Admin-Add

Admin-Remove

Admin Activate

Transaction Completed

(Active)

Force-Deactivate

Transaction Completed

(Inactive)

Admin Deactivate

Figure 1- LB Members State Transition Logic

Error: Reference source not found describes various states of the Load Balancer member services, transitions between these states and the corresponding notifications.

The following administrative actions can be applied to a LB-member service:

1. Add
2. Remove
3. Activate
4. Deactivate
5. Force-deactivate

* A member can be added to a Load Balancer in an “Active” or an “Inactive” state. When adding a member in an “Active” state, it is implicitely added in an “Inactive” state and transitioned to an “Active” state as described below.
* An existing member can be removed from a Load Balancer. When removing a member in an “Active” state, it is implicitely transitioned to an “Inactive” state as described below.
* When activated (1), the node enters a transient phase, during which it is considered to be in a “Active Pending” state.

Throughout this phase, sessions are not load balanced towards this member, and the load balancing mechanism is updated in a way that allows adding the new member without disrupting active sessions. The exact nature of this phase is implementation specific, it can e.g. implement the following internal stages:

* + Identify the members that will be effected by adding a new member
  + Identify the corresponding sessions, and ensure that they will not be disrupted during the update (add member) transaction
  + Update the load balancing domain in a way that ensures that active sessions are not disrupted
* Once the transaction ends (2), the member is transitioned to an “Active” state, and a corresponding notification is issued.

Removing or deactivating an active LB member (3) changes its state to “Inactive Pending”. This is a transition phase, during which active sessions may still exist on the corresponding LB member. Throughout this phase the load balancing mechanism is updated in a way that allows removing the member without disrupting active sessions. The exact nature of this phase is implementation specific; it can e.g. implement the following internal stages:

* + Identify active sessions and ensure that they will not be disrupted during the update (remove member from LB domain) transaction
  + Update the load balancing domain so that no new sessions are load balanced to the deactivated member
  + Wait to the active sessions to expire
* Once the “Inactive Pending” transaction completes (4), the member state is changed to “Inactive”, and a corresponding notification is issued
* In some cases, members can be Force-deactivated without going through a draining phase (5). For example, if a member is detected to be unreachable (e.g. by a heartbeat mechanism), there is no need to go through a draining phase since the relevant sessions are considered to be broken regardless of the load balancing action. In such case, removing this member from the load balancing domain decreases the impact of an inactive member.  
  Another example is when a session does not expire within the expected timeframe

## L4LB SAI API

L4LB SAI definition relies on the existing [inc/sainexthopgroup.h](https://github.com/marian-pritsak/SAI/pull/1/files#diff-672b42e2040f1c7838146d75cd5dc3ed) API, with the additions and modifications described below.

This provides existing next-hop-group functionality, including:

* Defining VIP as a route towards the group. Restricting traffic to a specific port or a set of ports van be done using ACL rules (see [saiacl.h](https://github.com/marian-pritsak/SAI/blob/a3fd94df30f0d77c6d70e3af9c89b09bd1d6ea04/inc/saiacl.h))
* Defining next hop destination and format, including tunneling
* Counting per VS (next-hop-group) and per member service (next-hop-group member)
* Performing bulk attribute-set operations, such as modifying a state of multiple members at once, adding/removing multiple members etc., using sai\_bulk\_object\_\* APIs

### SAI Next Group Modifications

* Add a new type to the sai\_next\_hop\_group\_type\_t to identify a next hop group that is a L4 session aware load balancer
* Add the following enumerator to specify the LB member administrative states

|  |
| --- |
| /\*\*  \* @brief Attribute data for #SAI\_NEXT\_HOP\_GROUP\_MEMBER\_ATTR\_ADMIN\_STATE |
| \*/ |
| typedef enum \_sai\_next\_hop\_group\_member\_admin\_state\_t |
| { |
| /\*\* Active \*/ |
| SAI\_NEXT\_HOP\_GROUP\_MEMBER\_ADMIN\_STATE\_ACTIVE, |
|  |
| /\*\* Inactive. Graceful shutdown \*/ |
| SAI\_NEXT\_HOP\_GROUP\_MEMBER\_ADMIN\_STATE\_INACTIVE, |
|  |
| /\*\* Force inactive. Disruptive to the connections \*/ |
| SAI\_NEXT\_HOP\_GROUP\_MEMBER\_ADMIN\_STATE\_FORCE\_INACTIVE, |
|  |
| } sai\_next\_hop\_group\_member\_admin\_state\_t; |

* Add the following enumerator to specify the LB member operational states

|  |
| --- |
| /\*\*  \* @brief Attribute data for #SAI\_NEXT\_HOP\_GROUP\_MEMBER\_ATTR\_OPER\_STATE |
| \*/ |
| typedef enum \_sai\_next\_hop\_group\_member\_oper\_state\_t |
| { |
| /\*\* Inactive. Does not serve any connections and is safe to remove \*/ |
| SAI\_NEXT\_HOP\_GROUP\_MEMBER\_OPER\_STATE\_INACTIVE, |
|  |
| /\*\* Pending becoming active. The new connections are not yet |
| \* distributed to that member \*/ |
| SAI\_NEXT\_HOP\_GROUP\_MEMBER\_OPER\_STATE\_ACTIVE\_PENDING, |
|  |
| /\*\* Active. Included in the distribution of the new connections \*/ |
| SAI\_NEXT\_HOP\_GROUP\_MEMBER\_OPER\_STATE\_ACTIVE, |
|  |
| /\*\* Pending becoming inactive. The new connections are no longer |
| \* distributed to this member, but it still has some open connections, |
| \* so it's not yet safe to remove. \*/ |
| SAI\_NEXT\_HOP\_GROUP\_MEMBER\_OPER\_STATE\_INACTIVE\_PENDING, |
|  |
| } sai\_next\_hop\_group\_member\_oper\_state\_t;   * Add the data structure returned by a notification callback informing of the updated state of an LB member following a state transition  |  | | --- | | /\*\*  \* @brief Defines the operational status of the next hop group member | | \*/ | | typedef struct \_sai\_next\_hop\_group\_member\_oper\_state\_notification\_t | | { | | /\*\* | | \* @brief Next hop group member id. | | \* | | \* @objects SAI\_OBJECT\_TYPE\_NEXT\_HOP\_GROUP\_MEMBER | | \*/ | | sai\_object\_id\_t member\_id; | |  | | /\*\* Next hop group member operational status \*/ | | sai\_next\_hop\_group\_member\_oper\_state\_t member\_status; | |  | | } sai\_next\_hop\_group\_member\_oper\_state\_notification\_t; | |  | |

* Add an enumerator value to set the size of the Load Balancing hash table, add to the enumerator ‘sai\_next\_hop\_group\_attr\_t’:

/\*\*

|  |
| --- |
| \* @brief Number of hash buckets in the group |
| \* |
| \* @type sai\_uint32\_t |
| \* @flags MANDATORY\_ON\_CREATE | CREATE\_ONLY |
| \* @validonly SAI\_NEXT\_HOP\_GROUP\_ATTR\_TYPE == SAI\_NEXT\_HOP\_GROUP\_TYPE\_L4\_SESSION\_CONSISTENT |
| \*/ |
| SAI\_NEXT\_HOP\_GROUP\_ATTR\_NUM\_OF\_BUCKETS, |

* Add an enumerator value to the next hop group attributes list, sai\_next\_hop\_group\_member\_attr\_t:

|  |
| --- |
| /\*\*  \* @brief Member admin state. Allows for disabling a member without removing it. |
| \* |
| \* Should only be used if the type of owning group is SAI\_NEXT\_HOP\_GROUP\_TYPE\_L4\_SESSION\_CONSISTENT |
| \* |
| \* @type sai\_next\_hop\_group\_member\_admin\_state\_t |
| \* @flags MANDATORY\_ON\_CREATE | CREATE\_AND\_SET |
| \*/ |
| SAI\_NEXT\_HOP\_GROUP\_MEMBER\_ATTR\_ADMIN\_STATE, |
|  |
| /\*\* |
| \* @brief Member operational status. |
| \* |
| \* Should only be used if the type of owning group is SAI\_NEXT\_HOP\_GROUP\_TYPE\_L4\_SESSION\_CONSISTENT |
| \* |
| \* @type sai\_next\_hop\_group\_member\_oper\_state\_t |
| \* @flags READ\_ONLY |
| \*/ |
| SAI\_NEXT\_HOP\_GROUP\_MEMBER\_ATTR\_OPER\_STATE, |

* Add the notification callback informing of the updated state of an LB member following a state transition

|  |
| --- |
| /\*\*  \* @brief Next hop group member state change notification |
| \* |
| \* @count data[count] |
| \* |
| \* @param[in] count Number of notifications |
| \* @param[in] data Array of next hop group member operational status |
| \*/ |
| typedef void (\*sai\_next\_hop\_group\_member\_state\_change\_notification\_fn)( |
| \_In\_ uint32\_t count, |
| \_In\_ const sai\_next\_hop\_group\_member\_oper\_state\_notification\_t \*data); |
|  |

### SAI Switch Modifications

And the corresponding entry in the switch attributes sai\_switch\_attr\_t ([inc/saiswitch.h](https://github.com/marian-pritsak/SAI/pull/1/files" \l "diff-048b4a2106e6b4a072d5c388e078c273)):

|  |
| --- |
| /\*\*  \* @brief Next hop group member state change notification callback |
| \* function passed to the adapter. |
| \* |
| \* In case driver does not support this attribute, The Host adapter should poll |
| \* port status by SAI\_NEXT\_HOP\_GROUP\_MEMBER\_ATTR\_OPER\_STATUS. |
| \* |
| \* Use sai\_next\_hop\_group\_member\_state\_change\_notification\_fn |
| \* as notification function. |
| \* |
| \* @type sai\_pointer\_t sai\_next\_hop\_group\_member\_state\_change\_notification\_fn |
| \* @flags CREATE\_AND\_SET |
| \* @default NULL |
| \*/ |
| SAI\_SWITCH\_ATTR\_NEXT\_HOP\_MEMBER\_OPER\_STATE\_CHANGE\_NOTIFY, |
|  |