|  |
| --- |
|  |

SONiC warm reboot

**Contents**

[1 Overview 1](#_Toc513546267)

[2 Use cases 2](#_Toc513546268)

[2.1 In-Service Restart 2](#_Toc513546269)

[2.1.1 Un-Planned restart 2](#_Toc513546270)

[2.1.2 BGP docker restart 2](#_Toc513546271)

[2.1.3 SWSS docker restart 2](#_Toc513546272)

[2.1.4 Syncd docker restart 3](#_Toc513546273)

[2.1.5 Teamd docker restart 3](#_Toc513546274)

[2.2 In-Service Upgrade 3](#_Toc513546275)

[2.2.1 Case 1: without SAI api call change 3](#_Toc513546276)

[2.2.2 Case 2: with SAI api call change 3](#_Toc513546277)

[2.3 Cold restart fallback 3](#_Toc513546278)

[3 Proposal 1: Reconciliation at Orchagent 4](#_Toc513546279)

[3.1 Key steps 4](#_Toc513546280)

[3.2 Questions 4](#_Toc513546281)

[3.2.1 How syncd restores to the state of pre-shutdown 4](#_Toc513546282)

[3.2.2 How Orchagent manages data dependencies during state restore 4](#_Toc513546283)

[3.2.3 What is missing in Orchagent for it to restore to the state of pre-shutdown 4](#_Toc513546284)

[3.2.4 How Orchagent gets the OID information 5](#_Toc513546285)

[3.2.5 How to handle the cases of SAI api call change during restore phase. 5](#_Toc513546286)

[3.2.6 How to deal with missing of notification during the reboot window 6](#_Toc513546287)

[3.3 Requirements on LibSAI and ASIC 6](#_Toc513546288)

[3.4 Requirements on syncd 6](#_Toc513546289)

[3.5 Requirement on network applications and orch data 6](#_Toc513546290)

[3.5.1 General requirements 6](#_Toc513546291)

[3.5.2 Port 7](#_Toc513546292)

[3.5.3 Lag/teamd 7](#_Toc513546293)

[3.5.4 Fdb 7](#_Toc513546294)

[3.5.5 Arp 7](#_Toc513546295)

[3.5.6 Route 7](#_Toc513546296)

[3.5.7 Acl 7](#_Toc513546297)

[3.5.8 Buffer 7](#_Toc513546298)

[3.5.9 Qos 7](#_Toc513546299)

[3.5.10 … 7](#_Toc513546300)

[3.6 Summary 7](#_Toc513546301)

[3.7 Approach evaluation 7](#_Toc513546302)

[3.7.1 Advantages 7](#_Toc513546303)

[3.7.2 Concerns/Issues with this approach 7](#_Toc513546304)

[4 Proposal 2: Reconciliation at syncd 7](#_Toc513546305)

[4.1 The existing syncd INIT/APPLY view framework 7](#_Toc513546306)

[4.1.1 Invariants for view comparison 8](#_Toc513546307)

[4.1.2 View comparison logic 8](#_Toc513546308)

[4.2 Approach evaluation 8](#_Toc513546309)

[4.2.1 Advantages 8](#_Toc513546310)

[4.2.2 Concerns/Issues with this approach 8](#_Toc513546311)

# Overview

The goal of SONiC warm reboot is to be able restart and upgrade SONiC software without impacting the data plane.

Warm restart of each individual process/docker is also part of the goal. Except for syncd and database docker, all other network applications and dockers is desired to support un-planned warm restart.

For restart processing, SONiC may be roughly divided into three layers:

**Network applications and Orchagent**: Putting them together is because each application will experience similar flow for warm start. Take route as example, upon restart operation, network application BGP performs graceful restart and gets synced with latest routing state via interacting with peers, RouteOrch/fpmsyncd take the input from BGP and deal with any stale/new routes. RouteOrch becomes up to date and propagates changes down to syncd.

**Syncd**: syncd should dump ASICDB before restart, and restore to the same state as pre-reboot. The restore of SONiC syncd itself should not disturb the state of ASIC. It takes changes from Orchagent and pass them down to LibSAI/ASIC after necessary transformation.

**LibSAI/ASIC**: ASIC vendor needs to ensure the state of ASIC and libSAI restores to the same state as pre-reboot.

neighsyncd

neighorch

BGP/ fpmsyncd

routeorch

Intfmgr/intfsyncd

Portsyncd/vlanmgr/teamd

portorch

intforch

ASIC/LibSAI

SYNCD

# Use cases

## In-Service Restart

The mechanism of restarting a component without impact to service. This assumes that the software version of the component has not changed after the restart.

There could be data changes like new/stale route, port state change, fdb change during restart window.

### Un-Planned restart

It is desired for all network applications and orchagent to be able to handle unplanned restart, and restore gracefully. It is not a requirement on syncd and ASIC/LibSAI due to dependency on ASIC processing.

### BGP docker restart

After BGP docker restart, new routes may be learned from BGP peers and some routes which had been pushed down to APPDB and ASIC may be gone. The system should be able to clear the stale route from APPDB down to ASIC and program the new route.

### SWSS docker restart

After swss docker restart, all the port/LAG, vlan, interface, arp and route data should be restored from configDB, APPDB, Linux Kernel and other reliable sources.

There could be port state, ARP, FDB changes during the restart window, proper sync processing should be performed.

### Syncd docker restart

The restart of syncd docker should leave data plane intact. After restart, syncd resumes control of ASIC/LibSAI and communication with swss docker. All other functions which run in syncd docker should be restored too like flexcounter processing.

### Teamd docker restart

The restart of teamd docker should not cause link flapping or any traffic loss. All lags at data plane should remain the same.

## In-Service Upgrade

The mechanism of upgrading to a newer version of a component without impacting service.

### Case 1: without SAI api call change

There are software changes in network applications like BGP, neighsyncd, portsyncd and even orchagent, but the changes don’t have impact on the interface with syncd as to the organization of existing data (meta data and dependency graph).

There could be data changes like new/stale route, port state change, fdb change during restart window. All the processing for 2.1 In-Service Restart applies here too.

### Case 2: with SAI api call change

#### Case 2.1 attribute change with SET

New version of orchagent may cause SET api to use a different value for certain attribute compared with previous version. Or a new attribute SET will be called.

#### Case 2.2 Object change with REMOVE

Object that existed in previous version may be deleted by default in new software version.

#### Case 2.3 Object change with CREATE

Two scenarios:

##### case 2.3.1 New SAI object

This is the new object defined at SAI layer and CREATE call is triggered at orchagent in new version of software.

##### case 2.3.2 Old object in previous version to be replaced with new object in new software version

Ex. Object will be created with more or less attributes or different attribute value, or multiple instance objects will be replaced with an aggregated object.

This is the most complex scenario, all other objects which have dependency on the old object should be cleaned up properly if the old object is not a leaf object.

## Cold restart fallback

An option to do cold restart or warm restart through configuration for all dockers should be provided.

Upon failure of warm restart, fallback mechanism to cold restart should be available.

# Proposal 1: Reconciliation at Orchagent

## Key steps

1. Restore to original state
   1. LibSAI/ASIC is able to restore to the state of pre-reboot without interrupting upper layer.
   2. Syncd is able to restore to the state of pre-reboot without interrupting ASIC and upper layer.
2. Remove stale date and perform new update
   1. Syncd state is driven by Orchagent (with exception of FDB), once it is restored, no need to perform reconciliation by itself.
   2. Orchagent ensures idempotent operation at LibSaiRedis interface, it will not pass down any create/remove/set operations on objects that had been performed before.
   3. After that Orchagent works together with each application to gathers any delta, and performs create(new object), set, or remove(stale object) operations for the delat data.
   4. Syncd processes the request from Orchagent as in normal boot.
3. States of ASIC/LibSAI, syncd, orchagent and applications become up to date now.

## Questions

### How syncd restores to the state of pre-shutdown

In this approach syncd only needs to save and restore the mapping between object RID and VID.

### How Orchagent manages data dependencies during state restore

The constructor of each orchagent subroutine may work as normal startup.

Data in CONFIGDB probably should be applied first, then APPDB data dump and other data like fdb dump are loaded with swssconfig producerstate interface. This allows orchagent to build up various data structure while ensuring the data dependencies.

Each network application and orchagent subroutine handle the dependency accordingly, which means some operation may be delayed until all required objects are ready. The dependency check has been part of existing implementation anyway, but new issues may pop up with this new scenario.

### What is missing in Orchagent for it to restore to the state of pre-shutdown

Orchagent and application may get data from configDB and APPDB as normal startup, but to be able to in sync and communicate with syncd, it also needs OID for each object with key type of sai\_object\_id\_t.

|  |
| --- |
| typedef struct \_sai\_object\_key\_t  { |
| union \_object\_key { |
| **sai\_object\_id\_t object\_id;** |
| sai\_fdb\_entry\_t fdb\_entry; |
| sai\_neighbor\_entry\_t neighbor\_entry; |
| sai\_route\_entry\_t route\_entry; |
| sai\_mcast\_fdb\_entry\_t mcast\_fdb\_entry; |
| sai\_l2mc\_entry\_t l2mc\_entry; |
| sai\_ipmc\_entry\_t ipmc\_entry; |
| sai\_inseg\_entry\_t inseg\_entry;  } key; |
| } sai\_object\_key\_t; |
|  |

### How Orchagent gets the OID information

For SAI redis create operation of those objects with object key type of sai\_object\_id\_t, Orchagent must be able to fetch the exact same OID as before shutdown, otherwise it will be out of sync with syncd. But current Orchagent implementation save OID in running time data struct only.

For object ID previously fetched via sai redis get operation, the same method still works.

One possible solution is to save the mapping between OID and attr\_list at redis\_generic\_create(). This assumes that during restore, exact same attr\_list will be used for object create, so same OID may be found and returned. Constructore phase mapping will be handled in separate table due to the fact that the attributes for an object may be change later, but during restore the default attributes are used again.

The other more complex solution is to save OID in redis DB for the created objects in each orchagent sub-routine and restore from that. This is good for handling the case of multiple OIDs mapped to the same attributes list.

Virtual OID should not be necessary in this solution. But it doesn’t hurt either if the virtual OID layer is kept.

Idempotency is required for LibSaiRedis interface.

### How to handle the cases of SAI api call change during restore phase.

[Case 2.1 attribute change with SET](#_Case_2.1_attribute) : at the sai\_redis\_generic\_set layer, based on the object key, compare attribute value and apply the change directly down to syncd/libsai/ASIC.

[Case 2.2 Object change with REMOVE](#_Case_2.2_Object): at the sai\_redis\_gereric\_remove layer, if the object key found in restoreDB, apply remove SAI api call directly down to syncd/libsai/ASIC. Dependency has been guaranteed at orchagent.

[Case 2.3 Object change with CREATE](#_Case_2.3_Object):

[case 2.3.1 New SAI object](#_case_2.3.1_New): Just apply the SAI API create operation down to syncd/libsai/ASIC. Dependency has been guaranteed at orchagent. But if it is not a leaf object, there will be cascading effect on other objects which has dependency on it when being created, which will be handled in next used case scenario. If the new SAI object is only used as an attribute in SET call for other objects, it could be handled in [Case 2.1 attribute change with SET](#_Case_2.1_attribute_1).

[case 2.3.2 Old object in previous version to be replaced with new object in new software version](#_case_2.3.2_Old): If this is a leaf object like route entry, neighbor entry, or fdb entry, just add version specific logic to remove it and create the new one.

Otherwise if there are other objects which have to use this object as one of the attributes during create call, those objects should be deleted first before deleting this old object. Version specific logic is needed here.

### How to deal with missing of notification during the reboot window

Port/fdb may have new state notification during reboot window? Probably the corresponding orchagent subroutine should perform get operation for the objects?

## Requirements on LibSAI and ASIC

LibSAI and ASIC should be able to save all necessary state upon shutdown request with warm restart option.

Upon create\_switch() request, LibSAI/ASIC should restore to the exact state of pre-shutdown.

Data plane should not be affected during the whole restore process.

Once restore is finished, LibSAI/ASIC works in normal operation state, they are agnostic of any warm restart processing happening in upper layer.

It is desired to support idempotency for create/remove/set in LibSAI, but may not be absolutely necessary for warm reboot solution.

## Requirements on syncd

Syncd should be able to save all necessary state upon shutdown request with warm restart option.

At the restart syncd should restore to the exact state of pre-shutdown.

Once restore is finished, syncd works in normal operation state, it is agnostic of any warm restart processing happening in upper layer.

## Requirement on network applications and orch data

### General requirements

Each application should be able to restore to the state of pre-shutdown.

Orchagent must be able to save and restore OID for objects created by Orchagent and with object key type of sai\_object\_id\_t. Other objects not created by Orchagent may restore OID via syncd interfaces.

The orchagent sub-routine of each application could use existing normal constructor and producerstate/consumerstate handling flow to ensure dependency and populate internal data structure.

After state restore, each application and corresponding orchagent sub-routine should be able to remove any stale object/state and perform any needed create/set.

### Port

### Lag/teamd

### Fdb

### Arp

### Route

### Acl

### Buffer

### Qos

### …

## Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer | Restore | Reconciliation | Idempotency | Dependency management |
| Application/Orchagent | Y | Y | Y for LibSaiRedis interface | Y |
| Syncd | Y | N | Good to have | Good to have |
| LibSAI/ASIC | Y | N | Good to have | Good to have |

## Approach evaluation

### Advantages

* straightforward logic, simple to implement for most upgrade/restart cases.
* Layer/applications decoupled, easy to divide and conquer.
* Each docker self contained, is prepared for unplanned warm restart of swss process and other network applications.

### Concerns/Issues with this approach

* Orchagent software upgrade could be handy, especially for the cases of SAI object replace which requires Orchagent to have use-once code to handle in service upgrade.

# Proposal 2: Reconciliation at syncd

## The existing syncd INIT/APPLY view framework

Essentially there will be two views created for warm restart. The current view represents the ASIC state before shutdown, temp view represents the new intended ASIC state after restart.

Based on the SAI object data model, each view is a directed acyclic graph, all(?) objects are linked together.

### Invariants for view comparison

#### Switch internal objects discovered vis SAI get operation.

They include SAI\_OBJECT\_TYPE\_PORT, SAI\_OBJECT\_TYPE\_QUEUE, SAI\_OBJECT\_TYPE\_SCHEDULER\_GROUP, SAI\_OBJECT\_TYPE\_SCHEDULER\_GROUP and a few more.

It is assumed that the RID/VID for those objects keep the same.

**Question 1**: what if there is change with those discovered object after version change?

**Question 2**: what if some of the discovered objects got changed? Like dynamic port breakout case.

#### Configured attribute values like VLAN id, interface IP and etc.

There could be change to the configured value, those not being changed may work as invariants.

**Question 3**: could some virtual OIDs for created objects in tmp view coincidently match with object in current view, but the objects are different? matchOids().

### View comparison logic

Utilizing the meta data of object, with those invariants as anchor points, for each object in temp view, it starts as root of a tree and go down to all layer of children node until leaf to find best match.

If no match is found, the object in temp view should be created, it is object CREATE operation.

If best match is found, but there is attributes different between the object in temp view and current view, SET operation should be performed.

Exact match yields Temp VID to Current VID translation, which also paves the way for upper layer comparison.

All objects in current VIEW which have reference count 0 at the end should be deleted, REMOVE operation.

**Question 4**: how to handle two objects with exactly same attributes? Example: overlay loopback RIF and underlay loopback RIF. VRF and possibly some other object in same situation?

**Question 5:** New version of software call create() API with one extra attribute, how will that be handled? Old way of create() plus set() for the extra attribute, or delete the existing object then create a brand new one?

**Question 6:** findCurrentBestMatchForGenericObject(), the method looks dynamic. What we need is deterministic processing which matches exactly what orchagent will do (if same operation is to be done there instead), no new unnecessary REMOVE/SET/CREATE, how to guarantee that?

## Approach evaluation

### Advantages

* Much simpler warm restart processing for orchagent.
* Generic processing based on SAI object model.

### Concerns/Issues with this approach

* Highly complex logic in syncd
* Warm restart of upper layer applications closely coupled with syncd.
* Various corner cases from SAI object model and changes in SAI object model itself have to be handled.