VLAN in SONiC

Table of Contents

[Requirements 1](#_Toc482193967)

[Assumptions 2](#_Toc482193968)

[New config model 2](#_Toc482193969)

[SAI 1.0 and beyond 2](#_Toc482193970)

[Major design 2](#_Toc482193971)

[VLAN schema 2](#_Toc482193972)

[Objects dependency 4](#_Toc482193973)

[Orchagent 4](#_Toc482193974)

[SAI API dependency and syncd 6](#_Toc482193975)

[Config Mgr, config DB and Minigraph 7](#_Toc482193976)

[Command Line 8](#_Toc482193977)

[Startup order 8](#_Toc482193978)

[Linux platform update 8](#_Toc482193979)

[Debugging and testing support 8](#_Toc482193980)

[Appendices 8](#_Toc482193981)

[A. Linux vlan aware bridge and ip over vlan configuration example 9](#_Toc482193982)

[B. VLAN trunk support if done under current config model 11](#_Toc482193983)

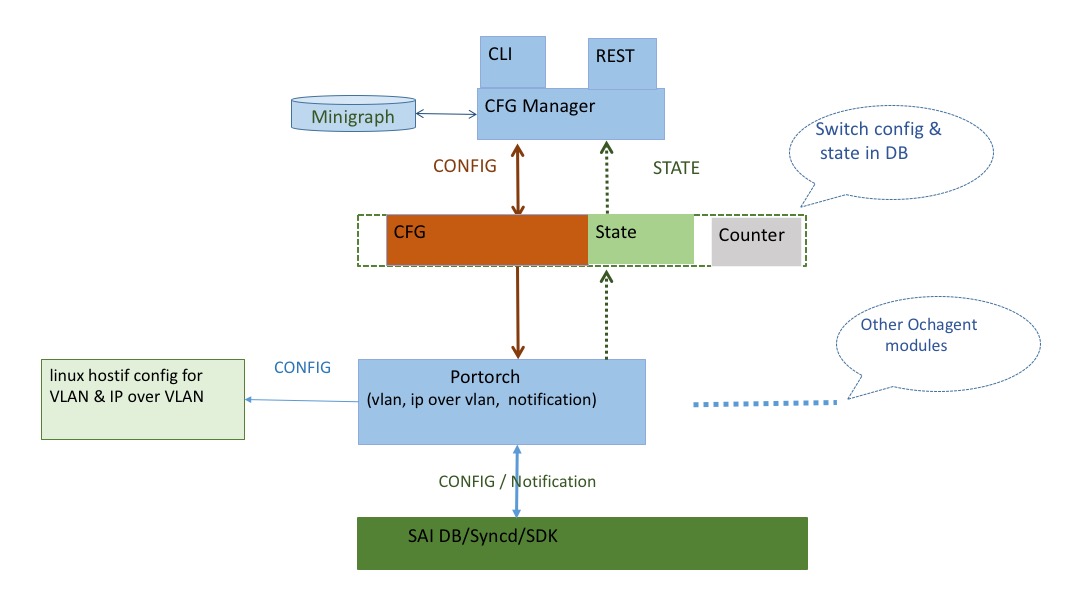
# Requirements

1. VLAN could be created in range of 1-4094
2. More than one VLAN could be created.
3. A VLAN could have members of physical switch ports or LAG or both types of them.
4. A physical port or LAG could be added to multiple VLANs in tagging mode of tagged.
5. A physical port or LAG could only be added to one VLAN in tagging mode of untagged.
6. Broadcast and unknown multicast packets could be prevented from flooding in a VLAN.
7. Unknown unicast packets could be prevented from flooding in a VLAN.
8. Up to 32 IP addresses could be configured on a VLAN interface.
9. It should be able to retrieve VLAN level stats.

# Assumptions

## New config model

One of the major assumptions for VLAN trunk design in this document is that the configuration of VLAN will be done through config manager based on the new config model.



Config Manager retrieves VLAN and IP over VLAN data from Minigraph or accepts the config request from CLI/Rest, then updates the objects into ConfigDB.

Portorch module will subscribe to the VLAN/Interface tables in ConfigDB. Upon any change in the corresponding ConfigDB tables, portorch updates Linux VLAN and IP config first, and proceeds to SAI objects update in SAI DB only when Linux part succeeded.

## SAI 1.0 and beyond

The 1Q bridge model in SAI 1.0 or newer version is needed for VLAN trunk. Readiness of SAI 1.0 in SONiC has been assumed.

# Major design

## VLAN schema

To follow the VLAN\_TABLE design in APP DB, but with minor extension for Broadcast/multicast DLF packets control and vlan description field.

###VLAN\_TABLE

;Defines VLANs and the interfaces which are members of the vlan

;Status: work in progress

key = VLAN\_TABLE:"vlan"vlanid ; DIGIT 0-4095 with prefix "Vlan"

admin\_status = "down" / "up" ; admin status

oper\_status = "down" / "up" ; operating status

mtu = 1\*4DIGIT ; MTU for the IP interface of the VLAN

**pkt\_control = "none" / "unicast\_miss"/"bum\_miss" ; default value as none**

**description = 1\*64VCHAR ; brief descript of this VLAN**

key = VLAN\_TABLE:vlanid:ifname ; physical port member of VLAN

tagging\_mode = "untagged" / "tagged" / "priority\_tagged" ; default value as untagged

When pkt\_control is set to unicast\_miss, unknown unicast packets will be prevented from being flooded in this VLAN domain. bum\_miss is to block all unknown broadcast/unicast/multicast packets. By default no flood control is enabled.

It may be difficult to put the attributes to SAI saivlan.h and have ASIC vendor to support them in time. As a workaround, we could use those switch level attributes:

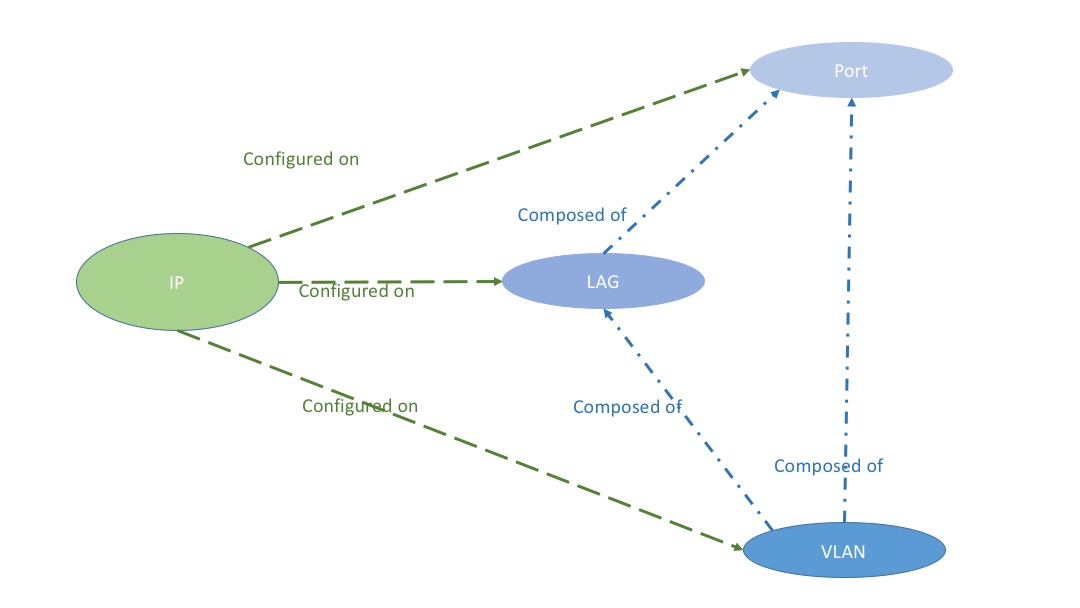
SAI\_SWITCH\_ATTR\_FDB\_UNICAST\_MISS\_PACKET\_ACTION,

SAI\_SWITCH\_ATTR\_FDB\_BROADCAST\_MISS\_PACKET\_ACTION,

SAI\_SWITCH\_ATTR\_FDB\_MULTICAST\_MISS\_PACKET\_ACTION,

Follow the example of SAI\_SWITCH\_ATTR\_SRC\_MAC\_ADDRESS for implementation in orchagent. Non-default attribute value could be fetched from minigraph upon orchagent docker start. Sonic-cfggen also needs some update for parsing support.

## Objects dependency



Port doesn’t have any config dependency.

LAG is composed of physical ports

Both Port and LAG could be member of VLAN

IP may be configured on VLAN, LAG or Port.

## Orchagent

VLAN TABLE, INTF\_TABLE for IP over VLAN need to be moved to new CONFIG DB, portorch/intforch subscribes to CONFIG DB for objects change notification.

portorch also updates operational state of VLAN to state part of the CONFIG DB (the exact behavior depends on the framework design for new CONFIG model).

Changes in Port class needs is necessary to accommodate the fact that a physical port or lag may be member of multiple VLANs and the VLAN member may be in tagged or untagged mode.

struct VlanMemberEntry

{

sai\_object\_id\_t vlan\_member\_id;

sai\_vlan\_tagging\_mode\_t vlan\_mode;

};

typedef map<sai\_vlan\_id\_t, VlanMemberEntry> Port\_Vlan\_members;

Since multiple types of ports reuse the same Port class, not all member variables apply to all port types, some cleanup or refactoring is preferred for clarity. Further update is needed after SAI 1.0 integration.

class Port

{

public:

…

std::string m\_alias;

Type m\_type;

int m\_index = 0; // PHY\_PORT: index

int m\_ifindex = 0;

union {

sai\_object\_id\_t m\_port\_id = 0;

sai\_vlan\_id\_t m\_vlan\_id = 0;

sai\_object\_id\_t m\_lag\_id = 0;

}

sai\_vlan\_id\_t m\_port\_vlan\_id = DEFAULT\_PORT\_VLAN\_ID; // Port VLAN ID

sai\_object\_id\_t m\_rif\_id = 0;

sai\_object\_id\_t m\_hif\_id = 0;

sai\_object\_id\_t m\_lag\_member\_id = 0; //used by physical port only

sai\_object\_id\_t m\_bridge\_port\_id;

Port\_Vlan\_members m\_vlan\_members; // Used by non-VLAN port

std::set<std::string> m\_members = set<std::string>(); //Used by vlan or lag

std::vector<sai\_object\_id\_t> m\_queue\_ids;

std::vector<sai\_object\_id\_t> m\_priority\_group\_ids;

};

At the beginning of each function addVlan()/removeVlan()/addVlanMember()/removeVlanMember(), Linux vlan aware bridge and ip link operation should be performed to ensure Linux environment is ready for control plane VLAN handling.

Operational state of DOWN for all VLAN member ports will bring VLAN interface LOWERLAYERDOWN state, Quagga zebra should have handled the state change properly for routing.

SAI Dot1Q bridge model will be used for the VLAN setup.

The processing for all other objects like PORT\_TABLE, LAG\_TABLE, and INTF\_TABLE for IP over port/LAG remains unchanged, which means the path for existing config model is used: Minigraph 🡪 Linux /etc/network/interfaces 🡪 portsyncd/infsyncd via netlink 🡪 APP DB 🡪 portorch/intforch.

## SAI API dependency and syncd

Part of those API and attributes required for VLAN trunk support are listed here:

// For adding/removing physical port/LAG into/from system default 1Q bridge

typedef sai\_status\_t (\*sai\_create\_bridge\_port\_fn)(

\_Out\_ sai\_object\_id\_t \*bridge\_port\_id,

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

\_In\_ uint32\_t attr\_count,

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

typedef sai\_status\_t (\*sai\_remove\_bridge\_port\_fn)(

\_In\_ sai\_object\_id\_t bridge\_port\_id);

// For VLAN create and remove

typedef sai\_status\_t (\*sai\_create\_vlan\_fn)(

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

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

\_In\_ uint32\_t attr\_count,

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

typedef sai\_status\_t (\*sai\_remove\_vlan\_fn)(

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

// For VLAN member create and remove

typedef sai\_status\_t (\*sai\_create\_vlan\_member\_fn)(

\_Out\_ sai\_object\_id\_t \*vlan\_member\_id,

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

\_In\_ uint32\_t attr\_count,

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

typedef sai\_status\_t (\*sai\_remove\_vlan\_member\_fn)(

\_In\_ sai\_object\_id\_t vlan\_member\_id);

// For setting SAI\_PORT\_ATTR\_PORT\_VLAN\_ID on port/LAG

typedef sai\_status\_t (\*sai\_set\_port\_attribute\_fn)(

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

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

typedef sai\_status\_t (\*sai\_set\_lag\_attribute\_fn)(

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

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

// Switch level attributes for broadcast packet control

SAI\_SWITCH\_ATTR\_FDB\_UNICAST\_MISS\_PACKET\_ACTION,

SAI\_SWITCH\_ATTR\_FDB\_BROADCAST\_MISS\_PACKET\_ACTION

SAI\_SWITCH\_ATTR\_FDB\_MULTICAST\_MISS\_PACKET\_ACTION

// VLAN level stats

typedef sai\_status\_t (\*sai\_get\_vlan\_stats\_fn)(

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

\_In\_ uint32\_t number\_of\_counters,

\_In\_ const sai\_vlan\_stat\_t \*counter\_ids,

\_Out\_ uint64\_t \*counters);

typedef sai\_status\_t (\*sai\_clear\_vlan\_stats\_fn)(

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

\_In\_ uint32\_t number\_of\_counters,

\_In\_ const sai\_vlan\_stat\_t \*counter\_ids);

PVID setting on LAG is not supported by SAI 1.0, as a workaround, PVID should be configured for each member of LAG.

syncd needs new support for VLAN stats collection. The counter data for VLAN should be injected to couter DB too. All these may be done in collectCountersThread syncd thread.

## Config Mgr, config DB and Minigraph

Upon docker start, config manager retrieves VLAN and IP over VLAN data from Minigraph, creates objects in config DB accordingly following VLAN\_TABLE and INTF\_TABLE schema.

Before the transition from existing APP DB to config DB is complete, config manager may need to access APP DB for VLAN member validity check like existence of LAG.

Minigraph and the sonic-config-engine should support the setting of tagging mode for VLAN members.

<VlanInterfaces>

<VlanInterface>

<Name>Vlan1000</Name>

<VlanID>1000</VlanID>

<VlanMembers>

<VlanMember>

<AttachTo>Ethernet96;Ethernet104</AttachTo>

<TaggingMode>untagged</TaggingMode>

</VlanMember>

<VlanMember>

<AttachTo>PortChannel01</AttachTo>

<TaggingMode>tagged</TaggingMode>

</VlanMember>

</VlanMembers>

</VlanInterface>

</VlanInterfaces>

VLAN related config in interface.j2 also shall be removed to avoid conflict.

## Command Line

Note: exact format of command line and output has dependency on the new config model framework.

[no] vlan <vlanid> [up|down]

[no] vlan <vlanid> member <port/lag name> [tagged|untagged]

[no] vlan ip address <IP> <netmask>

[no] vlan desc < description >

vlan show

## Startup order

Due to the dependency of VLAN config on LAG and port, and LAG and port configs are still going through the Linux netlink path, it is better for CONFIG manager to start after swss service to alleviate the race condition of VLAN config reaching orchagent before port and LAG.

## Linux platform update

Linux kernel module 8021q should be loaded by default at system startup.

To get around the partial result problem of Linux “bridge vlan” command, which is caused by the return value of “–EOPNOTSUPP” from igb\_ndo\_bridge\_getlink() function in IGB driver, one line patch is needed for Linux 3.16.36 kernel : http://elixir.free-electrons.com/linux/v4.5/source/net/core/rtnetlink.c#L3134

## Debugging and testing support

For Linux vlan setup and status, iproute2 suite commands should be good enough after the “–EOPNOTSUPP” fix.

Redis commands are nice approach for checking vlan data in CONFIG DB and SAI DB.

ASIC debugging command like bcmcmd is good for checking ASIC config and stats.

portstat utlity should be expanded for VLAN interface support.

A set of JSON files for practicing VLAN unit test against CONFIG DB should be prepared.

# Appendices

## Linux vlan aware bridge and ip over vlan configuration example

ip link del brt1

ip link add brt1 up type bridge

echo 1 > /sys/class/net/brt1/bridge/vlan\_filtering

bridge vlan del vid 1 dev brt1 self

bridge vlan add vid 1001 dev brt1 self pvid untagged

bridge vlan add vid 1002 dev brt1 self

bridge vlan add vid 1003 dev brt1 self

bridge vlan add vid 1004 dev brt1 self

bridge vlan add vid 1005 dev brt1 self

bridge vlan add vid 1006 dev brt1 self

ip link set eth2 master brt1

bridge vlan del vid 1 dev eth2

bridge vlan add vid 1004 dev eth2

bridge vlan add vid 1005 dev eth2

bridge vlan add vid 1006 dev eth2

bridge vlan add vid 1001 dev eth2 pvid untagged

ip link set eth1 master brt1

bridge vlan del vid 1 dev eth1

bridge vlan add vid 1004 dev eth1

bridge vlan add vid 1005 dev eth1

bridge vlan add vid 1006 dev eth1

bridge vlan add vid 1002 dev eth1

bridge vlan add vid 1003 dev eth1

bridge vlan add vid 1001 dev eth1 pvid untagged

ip link add link brt1 name Vlan1003 type vlan id 1003

ip address add 192.168.103.3/24 dev Vlan1003

ip address add 192.168.104.3/24 dev Vlan1003

root@debianhost2:/home/jipan# bridge vlan

port vlan ids

eth1 1001 PVID Egress Untagged

1002

1003

1004

1005

1006

eth2 1001 PVID Egress Untagged

1004

1005

1006

brt1 1001 PVID Egress Untagged

1002

1003

1004

1005

1006

root@debianhost2:/home/jipan# ip add show Vlan1003

36: Vlan1003@brt1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default

link/ether 52:54:00:00:bd:3f brd ff:ff:ff:ff:ff:ff

inet 192.168.103.3/24 scope global Vlan1003

valid\_lft forever preferred\_lft forever

inet 192.168.104.3/24 scope global Vlan1003

valid\_lft forever preferred\_lft forever

## 

## VLAN trunk support if done under current config model

