VLAN in SONiC

Table of Contents

[Requirements 1](#_Toc484988182)

[Assumptions 2](#_Toc484988183)

[New config model 2](#_Toc484988184)

[SAI 1.0 and beyond 3](#_Toc484988185)

[Limitations 3](#_Toc484988186)

[Major design 3](#_Toc484988187)

[Data schema update 3](#_Toc484988188)

[ConfigDB VLAN schema 3](#_Toc484988189)

[APPDB VLAN schema 4](#_Toc484988190)

[PORT\_TABLE schema 5](#_Toc484988191)

[LAG\_TABLE schema 5](#_Toc484988192)

[Objects dependency 6](#_Toc484988193)

[Portsyncd and intfsyncd 7](#_Toc484988194)

[Orchagent 7](#_Toc484988195)

[SAI API dependency and syncd 8](#_Toc484988196)

[saibridge.h 8](#_Toc484988197)

[saivlan.h 9](#_Toc484988198)

[saiport.h 10](#_Toc484988199)

[sailag.h 10](#_Toc484988200)

[saiswitch.h 10](#_Toc484988201)

[Config Mgr, config DB and Minigraph 11](#_Toc484988202)

[Command Lines 12](#_Toc484988203)

[Startup order 12](#_Toc484988204)

[Linux platform update 12](#_Toc484988205)

[Debugging and testing support 12](#_Toc484988206)

[Appendices 13](#_Toc484988207)

[A. Linux vlan aware bridge and ip over vlan configuration example 13](#_Toc484988208)

[B. VLAN trunk support if done under current config model 15](#_Toc484988209)

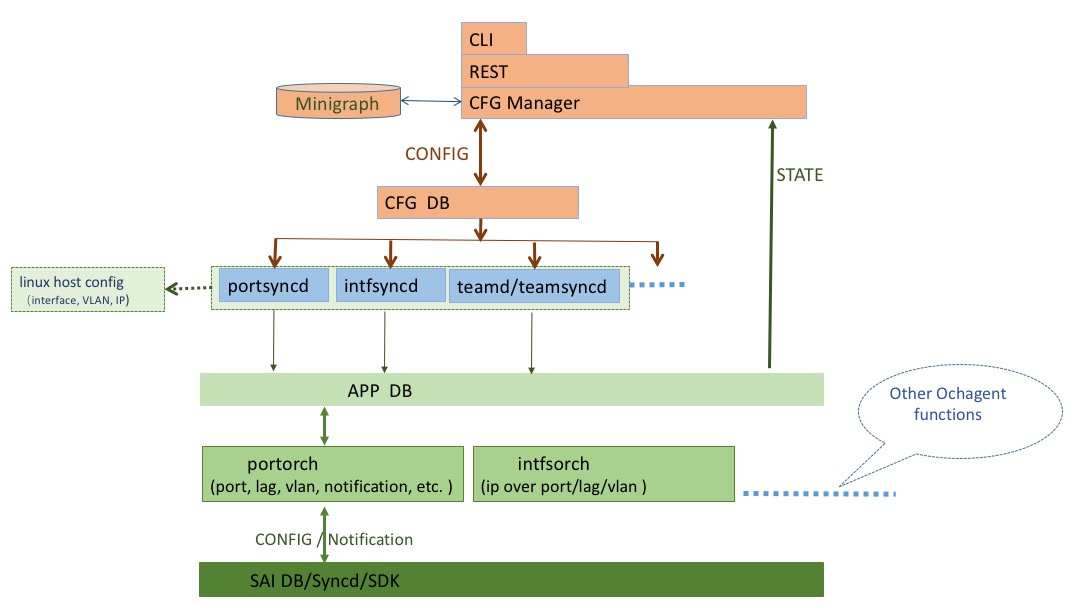
# Requirements

1. More than one 802.1Q VLAN could be created.
2. A VLAN could have members of physical ports or LAG or both types of them.
3. A physical port or LAG could be added to multiple VLANs in tagged mode.
4. A physical port or LAG could be added to one VLAN in untagged mode.
5. It shall be possible to config port VLAN ID (pvid) for individual physical port and LAG.
6. Up to 32 IP addresses could be configured on a VLAN interface.
7. Unknown unicast packets could be prevented from flooding in a VLAN.
8. Unknown multicast packets could be prevented from flooding in a VLAN.
9. Broadcast packets could be prevented from flooding in a VLAN.
10. It should be able to retrieve VLAN level stats.
11. By default VLAN interface is up regardless of member state. VLAN autostate may be enabled.

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

Portsyncd/intfsyncd modules subscribe to the VLAN/Interface tables in ConfigDB. For any change in the corresponding ConfigDB tables, portsyncd/intfsyncd perform Linux host config for VLAN and IP, upon success they continue updating the VLAN/Interface tables in APPDB.

Orchagent subscribes to VLAN/Interface tables in APPDB and is responsible for programming SAI DB.

Note that, VLAN table in APPDB is a superset of VLAN table in ConfigDB, it contains data for both configuration and state.

## 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.

# Limitations

As specified in the requirements section, only a subset of 802.1Q VLAN features are to be implemented.

Popular VLAN concepts like access, trunk, hybrid mode will not be implemented directly. Since they are just names for different combinations of VLAN tagging behavior for ingress/egress packets, special UI may be provided at external management system or at the enriched CLI wrapper layer based on user preference.

# Major design

## Data schema update

### ConfigDB VLAN schema

###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

autostate = “disabled”/ “enabled” ; default value as disabled

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

unicast\_miss\_flood = "true" / "false" ; default value as true

multicast\_miss\_flood = "true" / "false" ; default value as true

broadcast\_miss\_flood = "true" / "false" ; default value as true

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

config\_status\_code = 32HEXDIG ;

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

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

config\_status\_code = 32HEXDIG ;

Refer to https://github.com/opencomputeproject/SAI/blob/master/inc/saistatus.h for config\_status\_code, we intend to have a consistent view of status code within sonic.

### APPDB VLAN schema

To follow the VLAN\_TABLE design in APP DB, but with minor extension for Broadcast/multicast and 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

autostate = “disabled”/ “enabled” ; default value as disabled

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

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

unicast\_miss\_flood = "true" / "false" ; default value as true

multicast\_miss\_flood = "true" / "false" ; default value as true

broadcast\_miss\_flood = "true" / "false" ; default value as true

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

config\_status\_code = 32HEXDIG ;

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

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

config\_status\_code = 32HEXDIG ;

By default, unicast\_miss\_flood, multicast\_miss\_flood and broadcast\_miss\_flood all set to true, 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.

Refer to https://github.com/opencomputeproject/SAI/blob/master/inc/saistatus.h for config\_status\_code, we intend to have a consistent view of status code within sonic.

For VLAN interface, by default it stays in up status regardless member operational state. When autostate is enabled, VLAN interface goes down if no member is up operationally.

### PORT\_TABLE schema

###PORT\_TABLE

Stores information for physical switch ports managed by the switch chip. device\_names are defined in [port\_config.ini](../portsyncd/port\_config.ini). Ports to the CPU (ie: management port) and logical ports (loopback) are not declared in the PORT\_TABLE. See INTF\_TABLE.

;Defines layer 2 ports

;In SONiC, Data is loaded from configuration file by portsyncd

;Status: Mandatory

port\_table\_key = PORT\_TABLE:ifname ; ifname must be unique across PORT,INTF,VLAN,LAG TABLES

device\_name = 1\*64VCHAR ; must be unique across PORT,INTF,VLAN,LAG TABLES and must map to PORT\_TABLE.name

admin\_status = BIT ; is the port enabled (1) or disabled (0)

oper\_status = BIT ; physical status up (1) or down (0) of the link attached to this port

lanes = list of lanes ; (need format spec???)

ifname = 1\*64VCHAR ; name of the port, must be unique

mac = 12HEXDIG ;

;QOS Mappings

map\_dscp\_to\_tc = ref\_hash\_key\_reference

map\_tc\_to\_queue = ref\_hash\_key\_reference

;port VLAN ID

pvid = 1\*4DIGIT ; DIGIT 1-4095

### LAG\_TABLE schema

###LAG\_TABLE

;a logical, link aggregation group interface (802.3ad) made of one or more ports

;In SONiC, data is loaded by teamsyncd

;Status: work in progress

key = LAG\_TABLE:lagname ; logical 802.3ad LAG interface

minimum\_links = 1\*2DIGIT ; to be implemented

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

oper\_status = "down" / "up" ; Oper status (physical + protocol state)

mtu = 1\*4DIGIT ; MTU for this object

;port VLAN ID

pvid = 1\*4DIGIT ; DIGIT 1-4095

linkup

speed

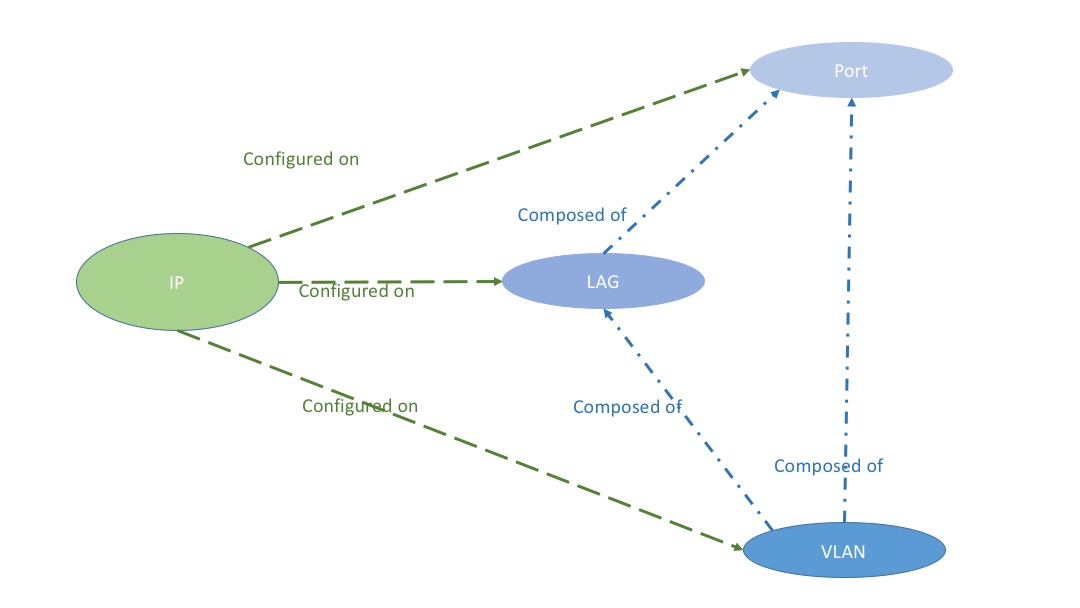
key = LAG\_TABLE:lagname:ifname ; physical port member of LAG, fk to PORT\_TABLE:ifname

status = "enabled" / "disabled" ; selected + distributing/collecting (802.3ad)

speed = ; set by LAG application, must match PORT\_TABLE.duplex

duplex = ; set by LAG application, must match PORT\_TABLE.duplex

## 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.

## Portsyncd and intfsyncd

Instead of listening to Linux netlink notications, portsyncd should watch VLAN\_TABLE in config DB for all VLAN operations. It takes care of Linux host config for VLAN host environment and is responsible for passing the request down to APP DB.

Intfsyncd does similar things for IP over VLAN config in INTF\_TABLE.

Linux vlan aware bridge and ip link operation is performed to ensure Linux environment is ready for control plane VLAN handling. SAI library should have ensured that trapped packets to host interfaces have proper VLAN tag, more specifically, host interface created by SAI library should not strip off VLAN tag of trapped packet once the corresponding physical port/LAG has been added to SAI Dot1Q bridge.

Updating config\_status\_code in corresponding configDB tables is the final step of config request processing.

## Orchagent

As done today, portorch/intforch subscribes to APPDB for objects change notification.

Changes in Port class 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

bool autostate; //Used by vlan, disabled by default

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;

};

Operational state of DOWN for all VLAN member ports will bring VLAN interface LOWERLAYERDOWN state. By default Linux and Quagga zebra treat the interface as UP for routing, with line protocol detection disabled. If autostate is enabled, VLAN interface also goes down with all members in down state.

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

It has been assumed that port and lag config would be moved to new config model before VLAN.

If VLAN function is to become the first user of new config model, then 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:

### saibridge.h

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

// All attributes in sai\_bridge\_port\_attr\_t which apply to 1Q bridge shall be supported.

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);

### saivlan.h

// For VLAN create and remove

// Following attributes shall be supported:

SAI\_VLAN\_ATTR\_VLAN\_ID,

SAI\_VLAN\_ATTR\_MEMBER\_LIST,

SAI\_VLAN\_ATTR\_MAX\_LEARNED\_ADDRESSES,

SAI\_VLAN\_ATTR\_LEARN\_DISABLE**,**

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

// All attributes in sai\_vlan\_member\_attr\_t shall be supported

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);

// All counters in sai\_vlan\_stat\_t shall be supported

What are missing but expected to be available in SAI VLAN interface

**SAI\_VLAN\_ATTR\_UNKNOWN\_UNICAST\_FLOOD\_GROUP,**

**SAI\_VLAN\_ATTR\_UNKNOWN\_MULTICAST\_FLOOD\_GROUP,**

**SAI\_VLAN\_ATTR\_BROADCAST\_FLOOD\_GROUP**

### saiport.h

// Following attributes shall be supported for VLAN

SAI\_PORT\_ATTR\_PORT\_VLAN\_ID

SAI\_PORT\_ATTR\_DEFAULT\_VLAN\_PRIORITY,

SAI\_PORT\_ATTR\_DROP\_UNTAGGED,

SAI\_PORT\_ATTR\_DROP\_TAGGED,

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);

// For port stats, following attributes shall be supported for VLAN

SAI\_PORT\_STAT\_IF\_IN\_VLAN\_DISCARDS,

### sailag.h

//What are missing but expected to be available in SAI LAG interface

SAI\_LAG\_ATTR\_PORT\_VLAN\_ID

SAI\_LAG\_ATTR\_DEFAULT\_VLAN\_PRIORITY,

SAI\_LAG\_ATTR\_DROP\_UNTAGGED,

SAI\_LAG\_ATTR\_DROP\_TAGGED,

### saiswitch.h

// 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

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 shall be removed to avoid conflict.

PVID for Port or lag PVID is a new item in minigraph

<PortChannelInterfaces>

<PortChannel>

<Name>PortChannel97</Name>

<AttachTo>Ethernet96</AttachTo>

**<Pvid>1000</Pvid>**

<SubInterface/>

</PortChannel>

</PortChannelInterfaces>

## Command Lines

Note: exact format of command line and output has dependency on the new config model framework. User may implement more advanced CLI format at CLI wrapper over REST interface.

[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

[no] port <portname> pvid <pvid>

[no] lag <lagname> pvid <pvid>

## Startup order

Proper startup order for different modules shall have been ensured in the new config model design.

If port and LAG config move to new config model after VLAN implementation, 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

Linux bridge floods unknown unicast, unknown multicast and broadcast in a bridge. New implementation is needed to provide options for disabling the flooding.

int [br\_handle\_frame\_finish](http://elixir.free-electrons.com/linux/v3.16.36/ident/br_handle_frame_finish)(struct [sk\_buff](http://elixir.free-electrons.com/linux/v3.16.36/ident/sk_buff) \*skb)

<http://elixir.free-electrons.com/linux/v3.16.36/source/net/bridge/br_input.c#L61> is the major f function to be touched.

## 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.

A similar utility like portstat could be implemented for VLAN interface support.

cfgmgrtest module for practicing VLAN unit test against CONFIG DB could be prepared.

# Appendices

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

brt1 is the name of bridge which has vlan\_filtering feature enabled in the example below:

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

