Network Interface Configuration on a Linux NOS

Roopa Prabhu— Cumulus Networks

Netdev 1.2, October 6th, 2016

Agenda

- Context and background
- Linux based NOS vs Linux as your NOS
- Network Interface management characteristics
- Network Interface management on a Linux NOS
- Search for a network interface manager for a Linux NOS
- ifupdown2
- Examples

Context and Background..

- Building a Linux OS distribution for routers and switches just like your server Linux distribution
- Leverage existing Linux ecosystem and tools
- Leverage existing automation tools: Make your network OS provisioning similar to your servers
- Goals of a network interface manager ?
  - Make network interface management painless and easy
  - Provision your network interfaces in the same way on servers and switches

Linux based NOS vs Linux as your NOS

Linux based NOS:

- Base Linux OS with vendor modifications
- Mostly closed boxes
- Proprietary API
- You almost never see the Linux behind it

Linux as your NOS:

- Linux as you see on servers + seamless hardware acceleration with switch asics
- Open boxes
- Open Linux networking API (Netlink)
- Leverage existing Linux ecosystem
- Automate like servers!

Characteristics of Linux network interface configuration

- Desktop and mobile operating system distributions:
  - Optimized for dynamic and changing networks
- Hypervisor and Container Operating system distributions:
  - Optimized for dynamic provisioning of networks for containers coming and going away
  - Networking parameters and attributes attached to a container or vm by orchestration tools
- Network Operating System distributions:
  - Mostly Static
  - Cookie cutter:
    - Eg: configure trunk vlans on all ports, configure all ports to 10G
  - Scale:
    - Large number of ports and large number of networking attributes
      - Eg: addresses, stp, igmp, vlans,

Our Goal for a network interface manager...

- Unify network interface management on servers and switches
  - All linux distributions use the same kernel netlink API or tools

- Keep it extensible with addon plugin modules for network configuration

- Optimize for a user-base using policies:
  - System policies:
    - Eg: Default system supported speed on an interface
  - User defined policies
    - Eg: vrf hooks, mtu

In search of a network interface manager for a Linux NOS...

- Requirements:
  - leverage existing Linux tools + API, Extensible, templatable
  - Already known to automation tools

- Started with Debian's ifupdown ….and currently at ifupdown2

- Ifupdown is a network interface manager on Debian (/etc/network/interfaces!)
  - https://packages.debian.org/jessie/admin/ifupdown

- Ifupdown2 is ifupdown optimized for a network operating system
  - https://github.com/CumulusNetworks/ifupdown2
  - https://packages.debian.org/sid/ifupdown2

ifupdown2

- Backward compatible with ifupdown interfaces format and commands
  - Continues to use **/etc/network/interfaces**
  - Understands interface dependencies
  - Pluggable architecture: add-on python modules for interface configuration
  - Interface configuration is templatable

 # ifupdown2 template example

 # configure 1000 vlan devices on
 # eth0
%for v in range(1, 1000):
auto eth0.${v}
iface eth0.${v}
%endfor

Next few slides ...

- Network interface configuration examples on a NOS
- ifupdown2 examples
- Default policies for a NOS where applicable

Physical ports and link attributes

Attributes:

- speed, duplex, autoneg setting using ethtool
- mtu, protodown using iproute2

Policies:

- System port manager policy to always set ‘autoneg on’ if port is 1G

/etc/network/interfaces example

```
auto swp1

Iface swp1

        link-speed 10000

        link-duplex full

        link-autoneg off

        mtu 9000

        hwaddress 00:02:0a:0b:0c:0d
```

L3 attributes

Attributes:

- address and static route configuration using iproute2 or direct netlink API to kernel

Policies:

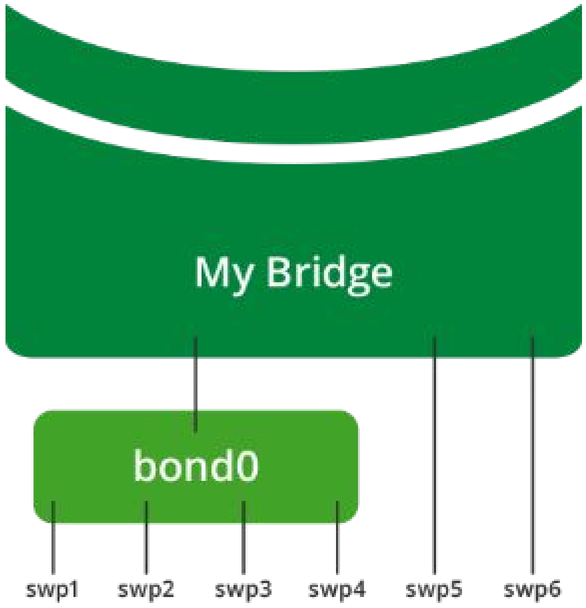
- policy to purge or not purge existing addresses (useful when address configuration is owned by multiple entities in the system)

/etc/network/interfaces example

```
auto swp1
iface swp1
        address 10.99.1.1/30
        post-up ip route add 10.1.2.0/24 via 10.99.1.2
```

Bonding or Link aggregation

- Bond creation and configuration using iproute2, sysfs and direct netlink API to kernel



/etc/network/interfaces example

auto bond0

iface bond0

bond-slaves glob swp1-3

bond-mode 802.3ad

auto bridge

iface bridge

bridge-ports swp1 bond0

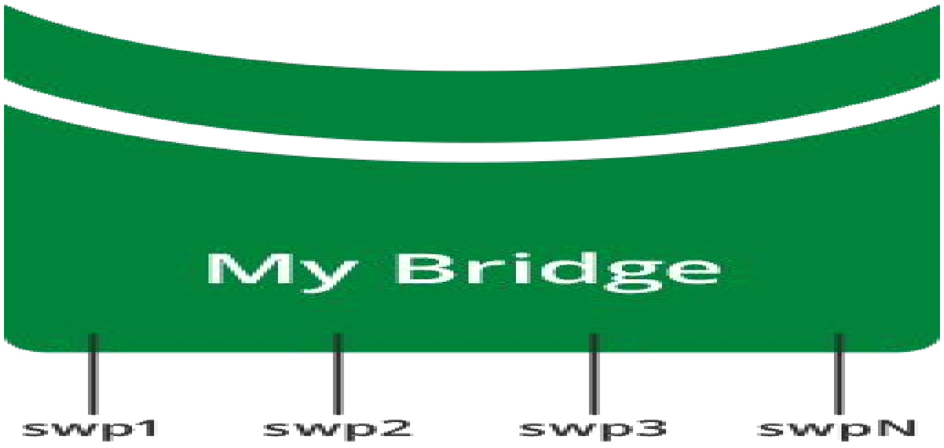
Bonding or Link aggregation: policy

System policy:

- restrict bond modes to network switch hardware link aggregation modes

Bridging

- Bridge attributes to indicate vlan filtering (vlan aware) bridge
- Easier ways to indicate range of ports



/etc/network/interfaces example

auto bridge

iface bridge

    bridge-vlan-aware yes

    bridge-ports glob swp1-3

    bridge-stp on

    bridge-vids 310 700 707 712 850 910

Bridging Contd

- Access port: sends and receives untagged ports (bridge-access)
- Trunk port: sends and receives tagged ports and able to switch multiple vlans (bridge-vids)
- Swp3 is a trunk uplink port inheriting all vlans from the bridge

/etc/network/interfaces example

```
auto swp1

iface swp1

    bridge-access 310

auto swp2

iface swp2

    bridge-vids 707 712 850

auto swp3

iface swp3
```

Bridging: policies

System policy:

- Prohibit addresses on a bridge port

Spanning tree protocol (STP) configuration

Linux kernel bridge driver stp

- Config using brctl, iproute2 or netlink

Stp in user space using mstpd

- Config using mstpctl

/etc/network/interfaces example

```
auto bridge
iface bridge
    bridge-vlan-aware yes
    bridge-ports swp1 swp2 swp3
    bridge-stp on
auto swp1
iface swp1
    mstpctl-bpduguard on
    mstpctl-portbpdufilter on
```

STP: policies

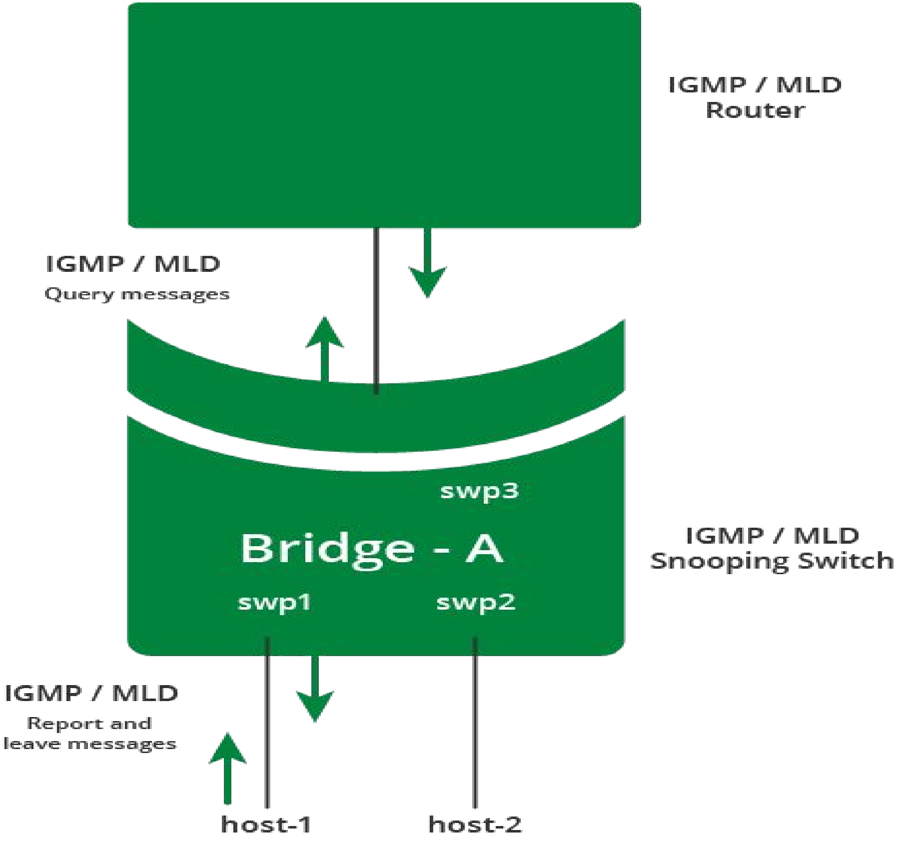
System policy:

- Default to STP bpdu off on vxlan bridge ports

IGMP snooping

Linux kernel bridge driver snoops igmp and mld packets

- Config using brctl, iproute2 or netlink



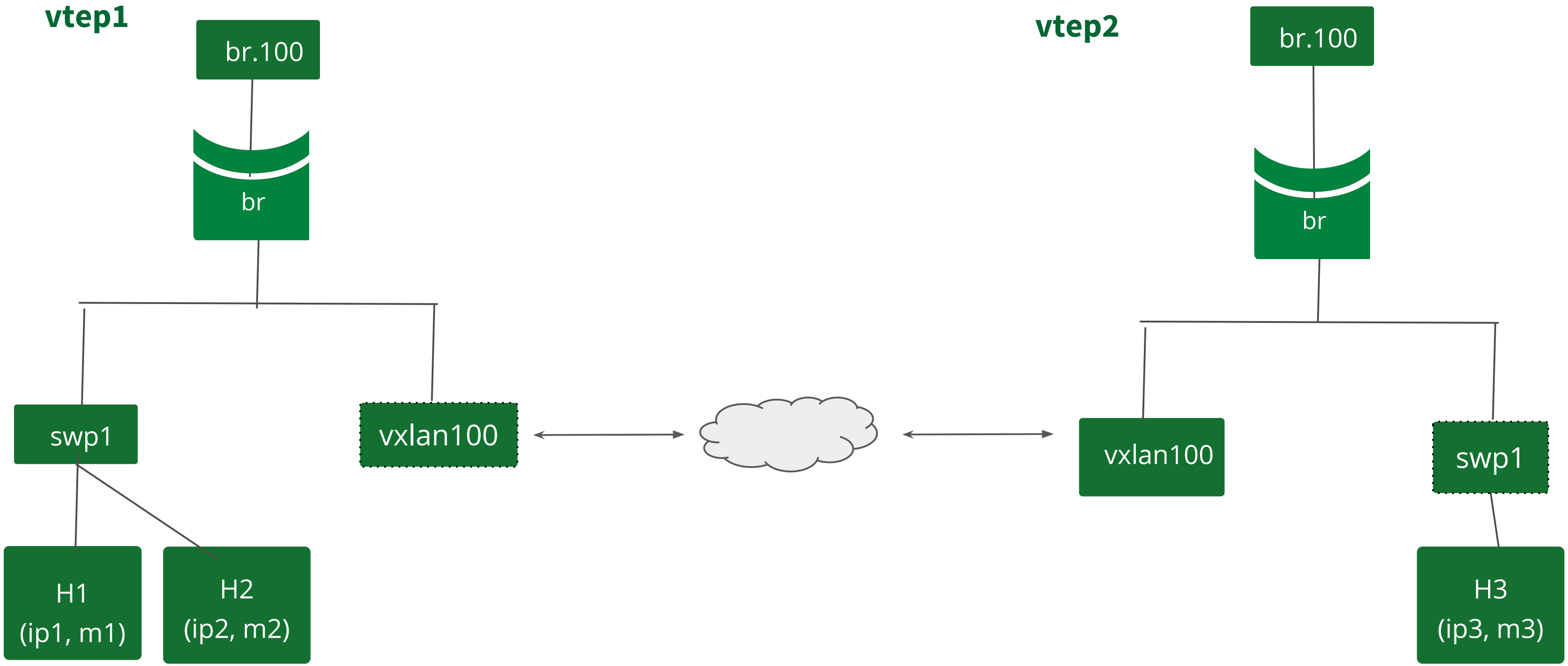
IGMP snooping contd

/etc/network/interfaces example

```
auto br0
iface br0 inet static
    bridge-ports swp1 swp2 swp3
    bridge-mcrouter 1
    bridge-mcsnoop 1
```

Vxlan Tunnel Endpoints (VTEPS)

vtep1, vtep2 : tors
H1, H2, H3: hosts



Vxlan Tunnel Endpoints (VTEPS) Contd

- Linux bridge to map end-host devices (vlan) to a vxlan segment (vni)

/etc/network/interfaces example

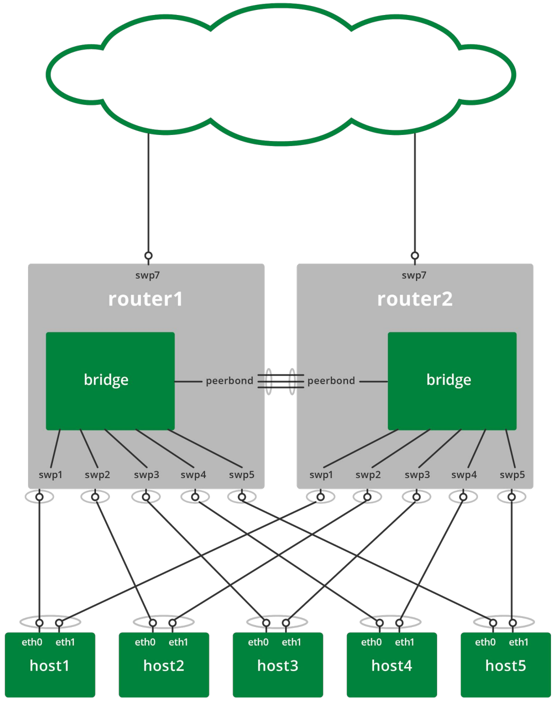
```
auto vxlan1000
iface vxlan1000
    vxlan-local-tunnelip 10.0.0.1
    vxlan-id 1000
    bridge-access 1000

auto bridge
iface bridge
    bridge-vlan-aware yes
    bridge-ports swp1 vxlan1000
    bridge-vids 1000
```

Vxlan Tunnel Endpoints (VTEPS): policies

System Policy:

- The vlan to vxlan mapping must be configured as a PVID on the vxlan bridge port

Virtual Redundant Router (VRR)



- VRR provides virtualized router redundancy
- A bridge connects all the local end-point devices
- A vlan subinterface on the bridge acts as a switched virtual interface or a layer3 interface for that vlan. This bridge vlan interface carries the original mac and ip for that vlan
- A Linux macvlan interface on top of the bridge vlan interface carries the virtual mac and ip
- The virtual mac and ip are common on both routers of a virtual redundant router pair

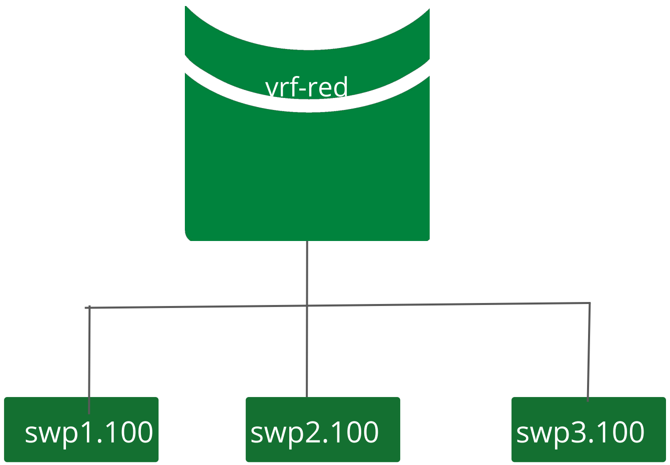
Virtual Redundant Router (VRR) Contd

/etc/network/interfaces example

```
auto bridge.100

iface bridge.100

    address 192.168.0.252/24

    address-virtual 00:00:5e:00:01:01
    192.168.0.254/24
```

```
auto bridge

iface bridge

    bridge-vlan-aware yes

    bridge-ports glob swp1-3
```

Virtual routing and forwarding (VRF)



- VRF allows for the presence of multiple independent routing tables working simultaneously on the same router or switch.
- This allows multiple network paths without the need for multiple switches.
- The VRF is represented as a layer3 master network device with its own associated routing table.
- Configuring a VRF involves creating a VRF master interface, allocating a routing table and enslaving interfaces to the VRF master device

Virtual routing and forwarding (VRF) contd

- **vrf-table** attribute
- **vrf** attribute under an interface to indicate vrf membership
- ifupdown2 maintains a vrf name and routing table id in **/etc/iproute2/rt_tables.d/ifupdown2_vrf_map.conf** file enabling easier references to vrf device and routing table by the vrf name

/etc/network/interfaces example

```
auto red
iface red
    vrf-table auto

auto swp1.100
iface swp1.100
    address 10.0.14.2/24
    vrf red
```

Virtual routing and forwarding (VRF): ifupdown2

/etc/network/interfaces example

```
auto blue

iface blue

        vrf-table auto


auto swp2.200

iface swp2.200

        address 10.0.15.2/24

        vrf blue
```

```
$cat /etc/iproute2/rt_tables.d/ifupdown2_vrf_map.conf

# This file is autogenerated by ifupdown2.

# It contains the vrf name to table mapping.

# Reserved table range 1001 1255

1001 red

1002 blue
```

Virtual routing and forwarding (VRF): policies

System policies:

- vrf table id reserved range: Reserving table id ranges helps a system administrator allocate kernel routing tables for various functions in the system.
- vrf max count: maps to hardware vrf limits
- vrf helper hook scripts: user provided scripts run at creation and deletion of a vrf
- vrf close sockets on down: close active sockets bound to the vrf device

Questions

?

Bringing the Linux Revolution to Networking

Thank You!