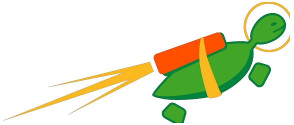
cumulus®



Scaling network interfaces on Linux

David Ahern, Nikolay Aleksandrov, Roopa Prabhu

Cumulus Networks

Feb 10th, 2016

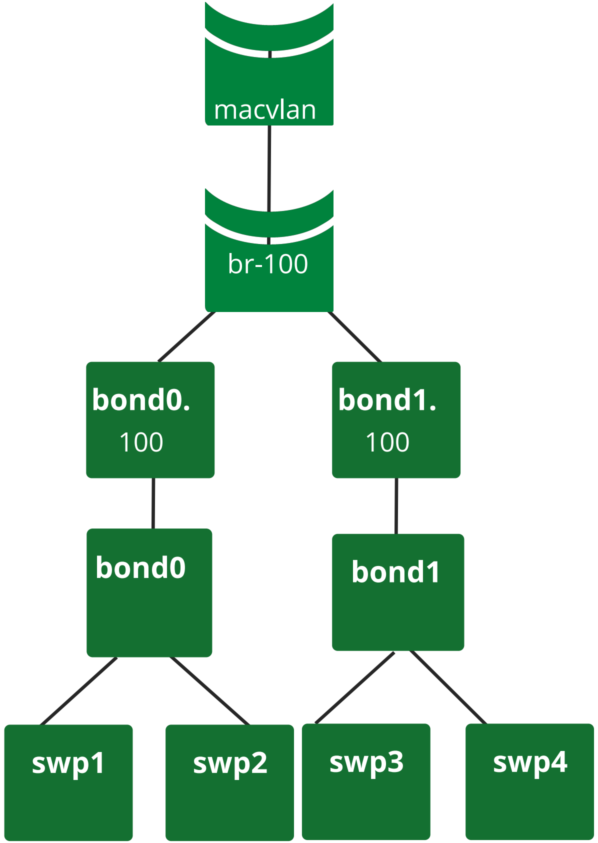
Agenda

- Examples of interface stacking
- net_devices and kernel memory impacts
- Userspace impacts

Bridging

- Ethernet bridges provide a means for hosts to communicate at layer 2
- Bridge members can be individual physical interfaces, bonds or logical interfaces that traverse an 802.1Q VLAN trunk

Network Interface Stacking (traditional bridge)



Worst Case:

128 physical interfaces (swp1, …, swp128)

+ 64 bonds

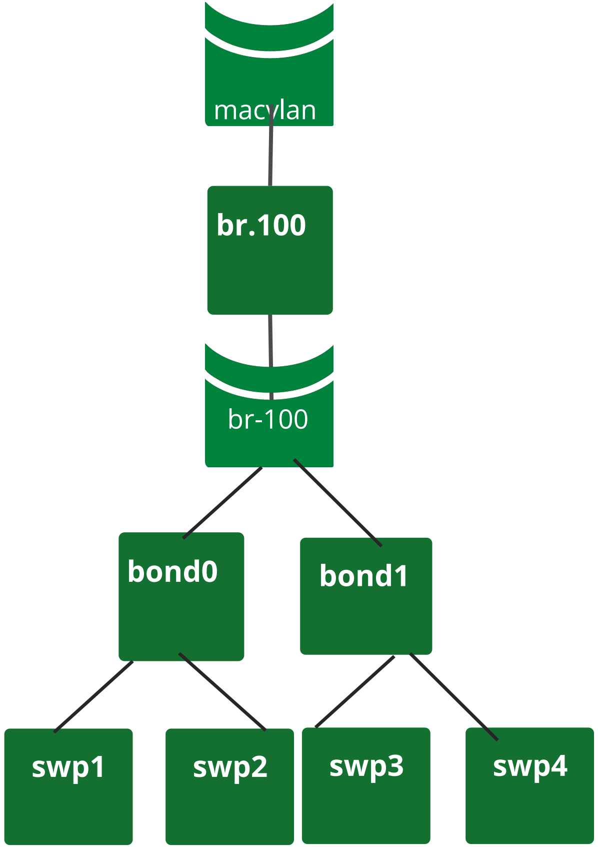
+ 64 * 4094 (bonds * vlan interfaces)

+ 4094 bridges

+ 4094 macvlan interfaces (1 per bridge)

270,396 network interfaces

Network Interface Stacking (vlan filtering bridge)



Worst Case:

128 physical interfaces (swp1, …, swp128)

+ 64 bonds

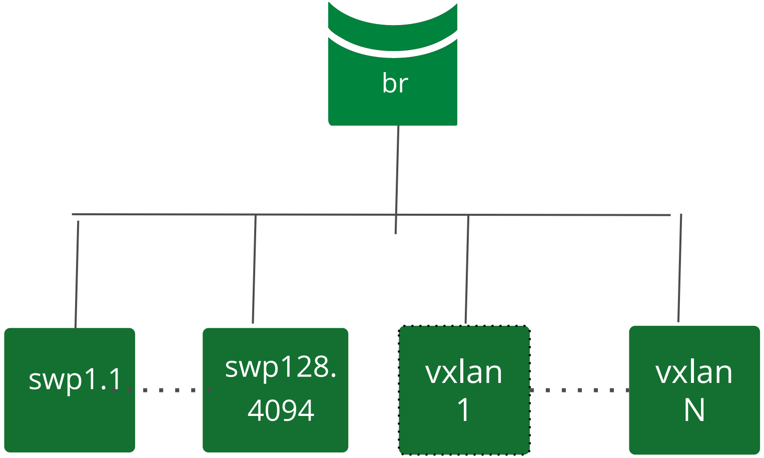
+ 1 bridge

+ 4094 vlan devices on bridge

+ 4094 macvlan interfaces

8,381 network interfaces

Network Interface Stacking (tunnels: vxlan l2 gateway)



forwarding between vlan and vxlan

Worst Case:

128 physical interface (swp1, …, swp128)

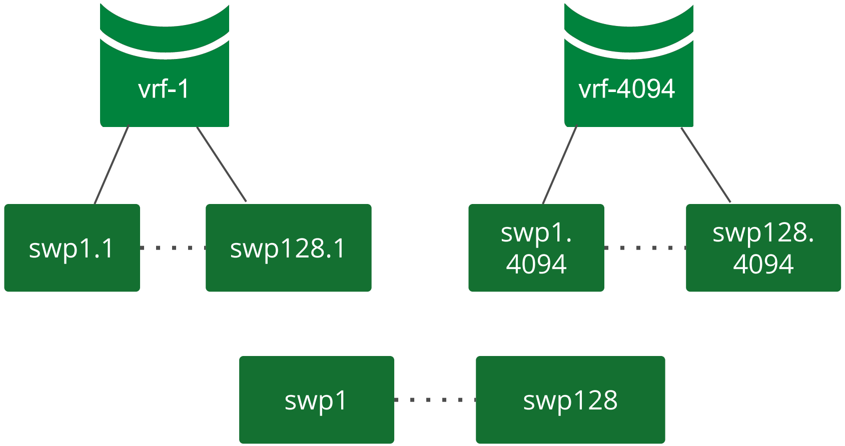
+ 128 * 4094 (vlans per interface)

+ 1 bridge

+ 2000 vxlan netdevs (e.g.,)

526,045 network interfaces

Network Interface Stacking (VRFs with VLANs)



Worst Case:

128 physical interfaces (swp1, …, swp128)

+ 128 * 4094 (vlans per interface)

+ 4094 VRFs

528,254 network interfaces

Basic modeling element in Linux networking stack

- ethernet interfaces
- vlan sub-interfaces
- bridges
- bonds
- vxlans
- tunnels
- vrfs

all represented as a netdev

Memory Use per Network Interface

Each 'ip link add' consumes at least 43k bytes

	requested	allocated
netdevice + hw address, queues	2,927	4,864
kobject + sysfs	14,321	14,568
IPv4 init	6,054	7,392
MPLS init	511	544
IPv6 init	12,637	16,576
Total	**36,450**	**43,944**

Memory Use for Stacking Cases

Use case	netdevs	netdev memory (MB)	private memory (MB)
traditional bridge	270,396	11,354	361
vlan filtering bridge	8,381	352	17
vxlan gateway	526,045	22,089	5

Options?

Lightweight tunnels (LWT)

- No netdevice created for tunnel endpoints; metadata on route entries

L2 only device

- skip L3 initializations (IPv4 and IPv6)

“Lightweight” netdevice

- drop the sysfs entries
- drop devconf (no sysctl entries) - “default” options are used

Lightweight Tunnels (LWT)

- Eliminate tunnel netdevices using LWT and flow based tunnels
- Attach tunnel attributes to routes instead of a tunnel netdevice
  - **VXLAN (single vxlan netdev vxlan0)**
    - ***ip route add 40.1.1.1/32 encap vxlan id 10 dst 50.1.1.2 dev vxlan0***
  - **MPLS (no netdevices carrying mpls tunnel attributes)**
    - ***ip route add 10.1.1.0/30 encap mpls 200 via inet 10.1.1.1 dev eth0***

Lightweight netdevice

Memory per device drops from ~45k range to ~13k

- 1/3rd the memory consumption per interface

Combine with L2 only

- Memory cost per net_device drops to 4,896 bytes

Lightweight netdevice

What’s the trade-off?

- no per-interface configuration (no /proc/sys/net entries)
  - defaults need to work for lwt-devs
- no sysfs entries -- impacts tools expecting run time stats and settings
  - tools need to use rtnetlink interface
- L2-only: No AF_INET + AF_INET6

UserSpace

Network interface managers

forwarding and routing protocols

- lldpd, bgp, ospf, stp

Redundancy/High availability

- clagd/mlagd, keepalived

monitoring and serviceability

- net-snmp

All react to changes with network interfaces

Admin

- Each netdev adds between 1196 (eth0) and 1556 (bridge) bytes
  - stats, address family-specific data (e.g, devconf)
- more user-kernel switches to retrieve data
  - Lot of data dumped to user
- ip has display filters - applied to data returned by kernel

Netlink notification overload

- Kernel notifies user space of changes in network interface attributes/network-protocol-states
- At scale, user space is constantly stormed with notifications
- Possible Solutions:
  - Reduce number of netdevices
  - more granular netlink notification filters

Example: fdb local mac problem:

- vlan filtering bridge: 128 ports * 4094 vlans
- fdb size explosion: 524032 entries, ~36 MB of memory
- per-fdb entry notifications

Options?

Kernel side filters

- Reduce data pushed to userspace

Which devices are collected:

- Option to return only devices enslaved to a given master device

  e.g., ip link show master br1

- Option to return only devices of a specific type

  e.g., ip link show type vrf

Options?

Reduce data per device

- Option to not send devconf - 696 bytes for IPv4 and IPv6
- Option to not send statistics - 284 bytes
- Combined stats + devconf are 980 of the 1200+ bytes

Example

615 network interfaces

- ethernet interfaces, vlans, bonds, bridges, macvlans, vrfs

ip link show

- regardless of filter (type, master, brief) requires 58 recvmsg calls, pulls in 888,512 bytes

with kernel side filtering

- ip link show type bond -- 7 recvmsg calls, 70,556 bytes
- ip link show type vrf -- 3 recvmsg calls, 3,716 bytes

Network Interface configuration at scale

- Mostly cookie-cutter interface configurations. eg: replicate the interface stack for all vlans
- flat files with large number of interfaces is cumbersome to manage
- Shrink network interface configuration specification: templatize

Example python mako template to create bridges for vlan 1000 to 1100 in ifupdown2:

```
%for v in range(1000,1100):
auto br-${v}
iface br-${v} inet static
   bridge-ports glob swp1-6.${v}
   bridge-stp on
%endfor
```

Questions

cumulus®

?

cumulus®

Bringing the Linux Revolution to Networking

Thank You!