Virtual Routing and Forwarding (VRF)

The VRF Device

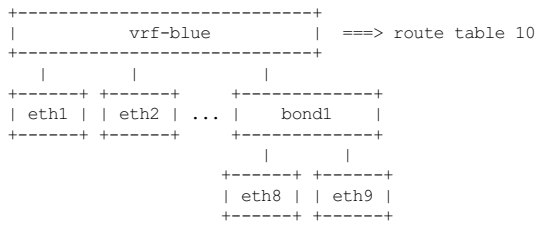
The VRF device combined with ip rules provides the ability to create virtual routing and forwarding domains (aka VRFs, VRF-lite to be specific) in the Linux network stack. One use case is the multi-tenancy problem where each tenant has their own unique routing tables and in the very least need different default gateways.

Processes can be "VRF aware" by binding a socket to the VRF device. Packets through the socket then use the routing table associated with the VRF device. An important feature of the VRF device implementation is that it impacts only Layer 3 and above so L2 tools (e.g., LLDP) are not affected (ie., they do not need to be run in each VRF). The design also allows the use of higher priority ip rules (Policy Based Routing, PBR) to take precedence over the VRF device rules directing specific traffic as desired.

In addition, VRF devices allow VRFs to be nested within namespaces. For example network namespaces provide separation of network interfaces at the device layer, VLANs on the interfaces within a namespace provide L2 separation and then VRF devices provide L3 separation.

Design

A VRF device is created with an associated route table. Network interfaces are then enslaved to a VRF device:



Packets received on an enslaved device and are switched to the VRF device in the IPv4 and IPv6 processing stacks giving the impression that packets flow through the VRF device. Similarly on egress routing rules are used to send packets to the VRF device driver before getting sent out the actual interface. This allows tcpdump on a VRF device to capture all packets into and out of the VRF as a whole[1]. Similarly, netfilter[2] and tc rules can be applied using the VRF device to specify rules that apply to the VRF domain as a whole.

[1] Packets in the forwarded state do not flow through the device, so those packets are not seen by tcpdump. Will revisit this limitation in a future release.

[2] Iptables on ingress supports PREROUTING with skb->dev set to the real ingress device and both INPUT and PREROUTING rules with skb->dev set to the VRF device. For egress POSTROUTING and OUTPUT rules can be written using either the VRF device or real egress device.

Setup

1. VRF device is created with an association to a FIB table. e.g.,:

   ```
   ip link add vrf-blue type vrf table 10
   ip link set dev vrf-blue up
   ```

2. An l3mdev FIB rule directs lookups to the table associated with the device. A single l3mdev rule is sufficient for all VRFs. The VRF device adds the l3mdev rule for IPv4 and IPv6 when the first device is created with a default preference of 1000. Users may delete the rule if desired and add with a different priority or install per-VRF rules.

   Prior to the v4.8 kernel iif and oif rules are needed for each VRF device:

   ```
   ip ru add oif vrf-blue table 10
   ip ru add iif vrf-blue table 10
   ```

3. Set the default route for the table (and hence default route for the VRF):

   ```
   ip route add table 10 unreachable default metric 4278198272
   ```

   This high metric value ensures that the default unreachable route can be overridden by a routing protocol suite. FRRouting interprets kernel metrics as a combined admin distance (upper byte) and priority (lower 3 bytes). Thus the above metric translates to [255/8192].

4. Enslave L3 interfaces to a VRF device:

   ```
   ip link set dev eth1 master vrf-blue
   ```

   Local and connected routes for enslaved devices are automatically moved to the table associated with VRF device. Any additional routes depending on the enslaved device are dropped and will need to be reinserted to the VRF FIB table following the enslavement.

   The IPv6 sysctl option keep_addr_on_down can be enabled to keep IPv6 global addresses as VRF enslavement changes:

   ```
   sysctl -w net.ipv6.conf.all.keep_addr_on_down=1
   ```

5. Additional VRF routes are added to associated table:

   ```
   ip route add table 10 ...
   ```

Applications

Applications that are to work within a VRF need to bind their socket to the VRF device:

```
setsockopt(sd, SOL_SOCKET, SO_BINDTODEVICE, dev, strlen(dev)+1);
```

or to specify the output device using cmsg and IP_PKTINFO.

By default the scope of the port bindings for unbound sockets is limited to the default VRF. That is, it will not be matched by packets

arriving on interfaces enslaved to an l3mdev and processes may bind to the same port if they bind to an l3mdev.

TCP & UDP services running in the default VRF context (ie., not bound to any VRF device) can work across all VRF domains by enabling the tcp_l3mdev_accept and udp_l3mdev_accept sysctl options:

```
sysctl -w net.ipv4.tcp_l3mdev_accept=1
sysctl -w net.ipv4.udp_l3mdev_accept=1
```

These options are disabled by default so that a socket in a VRF is only selected for packets in that VRF. There is a similar option for RAW sockets, which is enabled by default for reasons of backwards compatibility. This is so as to specify the output device with cmsg and IP_PKTINFO, but using a socket not bound to the corresponding VRF. This allows e.g. older ping implementations to be run with specifying the device but without executing it in the VRF. This option can be disabled so that packets received in a VRF context are only handled by a raw socket bound to the VRF, and packets in the default VRF are only handled by a socket not bound to any VRF:

```
sysctl -w net.ipv4.raw_l3mdev_accept=0
```

netfilter rules on the VRF device can be used to limit access to services running in the default VRF context as well.

Using VRF-aware applications (applications which simultaneously create sockets outside and inside VRFs) in conjunction with `net.ipv4.tcp_l3mdev_accept=1` is possible but may lead to problems in some situations. With that sysctl value, it is unspecified which listening socket will be selected to handle connections for VRF traffic; ie. either a socket bound to the VRF or an unbound socket may be used to accept new connections from a VRF. This somewhat unexpected behavior can lead to problems if sockets are configured with extra options (ex. TCP MD5 keys) with the expectation that VRF traffic will exclusively be handled by sockets bound to VRFs, as would be the case with `net.ipv4.tcp_l3mdev_accept=0`. Finally and as a reminder, regardless of which listening socket is selected, established sockets will be created in the VRF based on the ingress interface, as documented earlier.

Using iproute2 for VRFs

iproute2 supports the vrf keyword as of v4.7. For backwards compatibility this section lists both commands where appropriate -- with the vrf keyword and the older form without it.

1. Create a VRF

   To instantiate a VRF device and associate it with a table:

   ```
   $ ip link add dev NAME type vrf table ID
   ```

   As of v4.8 the kernel supports the l3mdev FIB rule where a single rule covers all VRFs. The l3mdev rule is created for IPv4 and IPv6 on first device create.

2. List VRFs

   To list VRFs that have been created:

   ```
   $ ip [-d] link show type vrf
     NOTE: The -d option is needed to show the table id
   ```

   For example:

   ```
   $ ip -d link show type vrf
   11: mgmt: <NOARP,MASTER,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
       link/ether 72:b3:ba:91:e2:24 brd ff:ff:ff:ff:ff:ff promiscuity 0
       vrf table 1 addrgenmode eui64
   12: red: <NOARP,MASTER,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
       link/ether b6:6f:6e:f6:da:73 brd ff:ff:ff:ff:ff:ff promiscuity 0
       vrf table 10 addrgenmode eui64
   13: blue: <NOARP,MASTER,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
       link/ether 36:62:e8:7d:bb:8c brd ff:ff:ff:ff:ff:ff promiscuity 0
       vrf table 66 addrgenmode eui64
   14: green: <NOARP,MASTER,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
       link/ether e6:28:b8:63:70:bb brd ff:ff:ff:ff:ff:ff promiscuity 0
       vrf table 81 addrgenmode eui64
   ```

   Or in brief output:

   ```
   $ ip -br link show type vrf
   mgmt         UP             72:b3:ba:91:e2:24 <NOARP,MASTER,UP,LOWER_UP>
   red          UP             b6:6f:6e:f6:da:73 <NOARP,MASTER,UP,LOWER_UP>
   blue         UP             36:62:e8:7d:bb:8c <NOARP,MASTER,UP,LOWER_UP>
   green        UP             e6:28:b8:63:70:bb <NOARP,MASTER,UP,LOWER_UP>
   ```

3. Assign a Network Interface to a VRF

   Network interfaces are assigned to a VRF by enslaving the netdevice to a VRF device:

   ```
   $ ip link set dev NAME master NAME
   ```

   On enslavement connected and local routes are automatically moved to the table associated with the VRF device.

   For example:

   ```
   $ ip link set dev eth0 master mgmt
   ```

4. Show Devices Assigned to a VRF

   To show devices that have been assigned to a specific VRF add the master option to the ip command:

   ```
   $ ip link show vrf NAME
   $ ip link show master NAME
   ```

   For example:

   ```
   $ ip link show vrf red
   3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master red state UP mode DEFAULT group de
       link/ether 02:00:00:00:02:02 brd ff:ff:ff:ff:ff:ff
   4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master red state UP mode DEFAULT group de
       link/ether 02:00:00:00:02:03 brd ff:ff:ff:ff:ff:ff
   7: eth5: <BROADCAST,MULTICAST> mtu 1500 qdisc noop master red state DOWN mode DEFAULT group default qlen 1000
       link/ether 02:00:00:00:02:06 brd ff:ff:ff:ff:ff:ff
   ```

Or using the brief output:

```
$ ip -br link show vrf red
eth1             UP             02:00:00:00:02:02 <BROADCAST,MULTICAST,UP,LOWER_UP>
eth2             UP             02:00:00:00:02:03 <BROADCAST,MULTICAST,UP,LOWER_UP>
eth5             DOWN           02:00:00:00:02:06 <BROADCAST,MULTICAST>
```

5. Show Neighbor Entries for a VRF

   To list neighbor entries associated with devices enslaved to a VRF device add the master option to the ip command:

```
$ ip [-6] neigh show vrf NAME
$ ip [-6] neigh show master NAME
```

   For example:

```
$  ip neigh show vrf red
10.2.1.254 dev eth1 lladdr a6:d9:c7:4f:06:23 REACHABLE
10.2.2.254 dev eth2 lladdr 5e:54:01:6a:ee:80 REACHABLE

$ ip -6 neigh show vrf red
2002:1::64 dev eth1 lladdr a6:d9:c7:4f:06:23 REACHABLE
```

6. Show Addresses for a VRF

   To show addresses for interfaces associated with a VRF add the master option to the ip command:

```
$ ip addr show vrf NAME
$ ip addr show master NAME
```

   For example:

```
$ ip addr show vrf red
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master red state UP group default qlen 100
    link/ether 02:00:00:00:02:02 brd ff:ff:ff:ff:ff:ff
    inet 10.2.1.2/24 brd 10.2.1.255 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 2002:1::2/120 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::ff:fe00:202/64 scope link
       valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master red state UP group default qlen 100
    link/ether 02:00:00:00:02:03 brd ff:ff:ff:ff:ff:ff
    inet 10.2.2.2/24 brd 10.2.2.255 scope global eth2
       valid_lft forever preferred_lft forever
    inet6 2002:2::2/120 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::ff:fe00:203/64 scope link
       valid_lft forever preferred_lft forever
7: eth5: <BROADCAST,MULTICAST> mtu 1500 qdisc noop master red state DOWN group default qlen 1000
    link/ether 02:00:00:00:02:06 brd ff:ff:ff:ff:ff:ff
```

   Or in brief format:

```
$ ip -br addr show vrf red
eth1             UP             10.2.1.2/24 2002:1::2/120 fe80::ff:fe00:202/64
eth2             UP             10.2.2.2/24 2002:2::2/120 fe80::ff:fe00:203/64
eth5             DOWN
```

7. Show Routes for a VRF

   To show routes for a VRF use the ip command to display the table associated with the VRF device:

```
$ ip [-6] route show vrf NAME
$ ip [-6] route show table ID
```

   For example:

```
$ ip route show vrf red
unreachable default  metric 4278198272
broadcast 10.2.1.0 dev eth1  proto kernel  scope link  src 10.2.1.2
10.2.1.0/24 dev eth1  proto kernel  scope link  src 10.2.1.2
local 10.2.1.2 dev eth1  proto kernel  scope host  src 10.2.1.2
broadcast 10.2.1.255 dev eth1  proto kernel  scope link  src 10.2.1.2
broadcast 10.2.2.0 dev eth2  proto kernel  scope link  src 10.2.2.2
10.2.2.0/24 dev eth2  proto kernel  scope link  src 10.2.2.2
local 10.2.2.2 dev eth2  proto kernel  scope host  src 10.2.2.2
broadcast 10.2.2.255 dev eth2  proto kernel  scope link  src 10.2.2.2

$ ip -6 route show vrf red
local 2002:1:: dev lo  proto none  metric 0  pref medium
local 2002:1::2 dev lo  proto none  metric 0  pref medium
2002:1::/120 dev eth1  proto kernel  metric 256  pref medium
local 2002:2:: dev lo  proto none  metric 0  pref medium
local 2002:2::2 dev lo  proto none  metric 0  pref medium
2002:2::/120 dev eth2  proto kernel  metric 256  pref medium
local fe80:: dev lo  proto none  metric 0  pref medium
local fe80:: dev lo  proto none  metric 0  pref medium
local fe80::ff:fe00:202 dev lo  proto none  metric 0  pref medium
local fe80::ff:fe00:203 dev lo  proto none  metric 0  pref medium
fe80::/64 dev eth1  proto kernel  metric 256  pref medium
fe80::/64 dev eth2  proto kernel  metric 256  pref medium
ff00::/8 dev red  metric 256  pref medium
ff00::/8 dev eth1  metric 256  pref medium
ff00::/8 dev eth2  metric 256  pref medium
unreachable default dev lo  metric 4278198272  error -101 pref medium
```

8. Route Lookup for a VRF

   A test route lookup can be done for a VRF:

```
$ ip [-6] route get vrf NAME ADDRESS
$ ip [-6] route get oif NAME ADDRESS
```

   For example:

```
$ ip route get 10.2.1.40 vrf red
10.2.1.40 dev eth1  table red  src 10.2.1.2
    cache

$ ip -6 route get 2002:1::32 vrf red
2002:1::32 from :: dev eth1  table red  proto kernel  src 2002:1::2  metric 256  pref medium
```

9. Removing Network Interface from a VRF

   Network interfaces are removed from a VRF by breaking the enslavement to the VRF device:

   ```
   $ ip link set dev NAME nomaster
   ```

   Connected routes are moved back to the default table and local entries are moved to the local table.

   For example:

   ```
   $ ip link set dev eth0 nomaster
   ```

Commands used in this example:

```
cat >> /etc/iproute2/rt_tables.d/vrf.conf <<EOF
1  mgmt
10 red
66 blue
81 green
EOF

function vrf_create
{
    VRF=$1
    TBID=$2

    # create VRF device
    ip link add ${VRF} type vrf table ${TBID}

    if [ "${VRF}" != "mgmt" ]; then
        ip route add table ${TBID} unreachable default metric 4278198272
    fi
    ip link set dev ${VRF} up
}

vrf_create mgmt 1
ip link set dev eth0 master mgmt

vrf_create red 10
ip link set dev eth1 master red
ip link set dev eth2 master red
ip link set dev eth5 master red

vrf_create blue 66
ip link set dev eth3 master blue

vrf_create green 81
ip link set dev eth4 master green


Interface addresses from /etc/network/interfaces:
auto eth0
iface eth0 inet static
      address 10.0.0.2
      netmask 255.255.255.0
      gateway 10.0.0.254

iface eth0 inet6 static
      address 2000:1::2
      netmask 120

auto eth1
iface eth1 inet static
      address 10.2.1.2
      netmask 255.255.255.0

iface eth1 inet6 static
      address 2002:1::2
      netmask 120

auto eth2
iface eth2 inet static
      address 10.2.2.2
      netmask 255.255.255.0

iface eth2 inet6 static
      address 2002:2::2
      netmask 120

auto eth3
iface eth3 inet static
      address 10.2.3.2
      netmask 255.255.255.0

iface eth3 inet6 static
      address 2002:3::2
      netmask 120

auto eth4
iface eth4 inet static
      address 10.2.4.2
      netmask 255.255.255.0

iface eth4 inet6 static
      address 2002:4::2
```

netmask 120