NET_FAILOVER

Overview

The net_failover driver provides an automated failover mechanism via APIs to create and destroy a failover master netdev and manages a primary and standby slave netdevs that get registered via the generic failover infrastructure.

The failover netdev acts a master device and controls 2 slave devices. The original paravirtual interface is registered as 'standby' slave netdev and a passthru/vf device with the same MAC gets registered as 'primary' slave netdev. Both 'standby' and 'failover' netdevs are associated with the same 'pci' device. The user accesses the network interface via 'failover' netdev. The 'failover' netdev chooses 'primary' netdev as default for transmits when it is available with link up and running.

This can be used by paravirtual drivers to enable an alternate low latency datapath. It also enables hypervisor controlled live migration of a VM with direct attached VF by failing over to the paravirtual datapath when the VF is unplugged.

virtio-net accelerated datapath: STANDBY mode

net_failover enables hypervisor controlled accelerated datapath to virtio-net enabled VMs in a transparent manner with no/minimal guest userspace changes.

To support this, the hypervisor needs to enable VIRTIO_NET_F_STANDBY feature on the virtio-net interface and assign the same MAC address to both virtio-net and VF interfaces.

Here is an example libvirt XML snippet that shows such configuration:

```
<interface type='network'>
  <mac address='52:54:00:00:12:53'/>
  <source network='enp66s0f0_br'/>
  <target dev='tap01'/>
  <model type='virtio'/>
  <driver name='vhost' queues='4'/>
  <link state='down'/>
  <teaming type='persistent'/>
  <alias name='ua-backup0'/>
</interface>
<interface type='hostdev' managed='yes'>
  <mac address='52:54:00:00:12:53'/>
  <source>
    <address type='pci' domain='0x0000' bus='0x42' slot='0x02' function='0x5'/>
  </source>
  <teaming type='transient' persistent='ua-backup0'/>
</interface>
```

In this configuration, the first device definition is for the virtio-net interface and this acts as the 'persistent' device indicating that this interface will always be plugged in. This is specified by the 'teaming' tag with required attribute type having value 'persistent'. The link state for the virtio-net device is set to 'down' to ensure that the 'failover' netdev prefers the VF passthrough device for normal communication. The virtio-net device will be brought UP during live migration to allow uninterrupted communication.

The second device definition is for the VF passthrough interface. Here the 'teaming' tag is provided with type 'transient' indicating that this device may periodically be unplugged. A second attribute - 'persistent' is provided and points to the alias name declared for the virtio-net device.

Booting a VM with the above configuration will result in the following 3 interfaces created in the VM:

```
4: ens10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 52:54:00:00:12:53 brd ff:ff:ff:ff:ff:ff
    inet 192.168.12.53/24 brd 192.168.12.255 scope global dynamic ens10
       valid_lft 42482sec preferred_lft 42482sec
    inet6 fe80::97d8:db2:8c10:b6d6/64 scope link
       valid_lft forever preferred_lft forever
5: ens10nsby: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master ens10 state DOWN group default qlen 10
    link/ether 52:54:00:00:12:53 brd ff:ff:ff:ff:ff:ff
7: ens11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ens10 state UP group default qlen 10
    link/ether 52:54:00:00:12:53 brd ff:ff:ff:ff:ff:ff
```

Here, ens10 is the 'failover' master interface, ens10nsby is the slave 'standby' virtio-net interface, and ens11 is the slave 'primary' VF passthrough interface.

One point to note here is that some user space network configuration daemons like systemd-networkd, ifupdown, etc, do not understand the 'net_failover' device; and on the first boot, the VM might end up with both 'failover' device and VF accquiring IP addresses (either same or different) from the DHCP server. This will result in lack of connectivity to the VM. So some tweaks might be needed to these network configuration daemons to make sure that an IP is received only on the 'failover' device.

Below is the patch snippet used with 'cloud-ifupdown-helper' script found on Debian cloud images:

::

    @@ -27,6 +27,8 @@ do_setup() {

local working="$cfgdir/.$INTERFACE" local final="$cfgdir/$INTERFACE"

- if [-d "/sys/class/net/${INTERFACE}/master"]; then exit 0; fi
- if ifup --no-act "$INTERFACE" > /dev/null 2>&1; then
  # interface is already known to ifupdown, no need to generate cfg log "Skipping configuration generation for $INTERFACE"

Live Migration of a VM with SR-IOV VF & virtio-net in STANDBY mode

net_failover also enables hypervisor controlled live migration to be supported with VMs that have direct attached SR-IOV VF devices by automatic failover to the paravirtual datapath when the VF is unplugged.

Here is a sample script that shows the steps to initiate live migration from the source hypervisor. Note: It is assumed that the VM is connected to a software bridge 'br0' which has a single VF attached to it along with the vnet device to the VM. This is not the VF that was passthrough'd to the VM (seen in the vf.xml file).

```
# cat vf.xml
<interface type='hostdev' managed='yes'>
  <mac address='52:54:00:00:12:53'/>
  <source>
    <address type='pci' domain='0x0000' bus='0x42' slot='0x02' function='0x5'/>
  </source>
  <teaming type='transient' persistent='ua-backup0'/>
</interface>

# Source Hypervisor migrate.sh
#!/bin/bash

DOMAIN=vm-01
PF=ens6np0
VF=ens6v1             # VF attached to the bridge.
VF_NUM=1
TAP_IF=vmtap01        # virtio-net interface in the VM.
VF_XML=vf.xml

MAC=52:54:00:00:12:53
ZERO_MAC=00:00:00:00:00:00

# Set the virtio-net interface up.
virsh domif-setlink $DOMAIN $TAP_IF up

# Remove the VF that was passthrough'd to the VM.
virsh detach-device --live --config $DOMAIN $VF_XML

ip link set $PF vf $VF_NUM mac $ZERO_MAC

# Add FDB entry for traffic to continue going to the VM via
# the VF -> br0 -> vnet interface path.
bridge fdb add $MAC dev $VF
bridge fdb add $MAC dev $TAP_IF master

# Migrate the VM
virsh migrate --live --persistent $DOMAIN qemu+ssh://$REMOTE_HOST/system

# Clean up FDB entries after migration completes.
bridge fdb del $MAC dev $VF
bridge fdb del $MAC dev $TAP_IF master
```

On the destination hypervisor, a shared bridge 'br0' is created before migration starts, and a VF from the destination PF is added to the bridge. Similarly an appropriate FDB entry is added.

The following script is executed on the destination hypervisor once migration completes, and it reattaches the VF to the VM and brings down the virtio-net interface.

::

    # reattach-vf.sh #!/bin/bash

    bridge fdb del 52:54:00:00:12:53 dev ens36v0 bridge fdb del 52:54:00:00:12:53 dev vmtap01 master virsh attach-device --config --live vm01 vf.xml virsh domif-setlink vm01 vmtap01 down