NETRONOME

Using
Agilio Open vSwitch
with
Virtio-Forwarder

Revision: 0.1
July 2018

REVISION HISTORY

Date	Revision	Description	Author
05-07-2018	0.1	Initial draft	Heinrich Kuhn

INTRODUCTION **3**

NETRONOME

ADDING VF's TO VIRTIO-FORWARDER **3**
VF CREATION 3
BINDING TO VFIO-PCI 3
ADDING VF's TO VIRTIO-FORWARDER **4**
VIRTIO-FORWARDER IN VHOSTUSER SERVER MODE **4**
STARTING VIRTIO-FORWARDER **5**
AGILIO OPEN VSWITCH CONFIGURATION **5**
GUEST CONFIGURATION **6**
TESTING CONNECTION **7**
LIVE MIGRATION **7**
LIVE MIGRATION USING LIBVIRT 7
LIVE MIGRATION IN A ORCHESTRATION SYSTEM 9
OPEN VSWITCH WITH NORMAL VIRTIO INTERFACES **9**

INTRODUCTION

This document is intended to provide a user the basic steps required to set up Virtio-Forwarder in conjunction with Agillio Open vSwitch.

Virtio-Forwarder (VIO4WD) is a userspace networking application that forwards bi-directional traffic between SR-IOV virtual functions (VFs) and Virtio networking devices in QEMU virtual machines. virtio-forwarder implements a virtio backend driver using the DPDK vhost-user library and services designated VFs by means of the DPDK poll mode driver (PMD) mechanism.

This document assumes that Agillio Open vSwitch has been set up successfully. For more information on setting up Agillio Open vSwitch please refer to the official User Guide (available in the Agilio OVS-TC download section of our support site)

This document also assumes that the setup steps for virtio-forwarder have been followed. Extensive documentation on virtio-forwarder can be found at http://virtio-forwarder.readthedocs.io/en/latest/

ADDING VF's TO VIRTIO-FORWARDER

VF CREATION

VF's for use with virtio-forwarder may be created using the following steps:

```
#Identify the Netronome smartNIC:
lspci -d19ee:

#Create 2 VF's
echo 2 > /sys/bus/pci/devices/<pci_addr>/sriov_numvfs
```

BINDING TO VFIO-PCI

The newly created VF's are bound to vfio-pci to allow them to be used by virtio-forwarder

```
# Example PCI address of VF
INTERFACE_PCI=0000:02:08.0
```

```
# Load the vfio-pci driver:
modprobe vfio-pci

#Unbind VF from nfp_netvf driver
echo ${INTERFACE_PCI} > /sys/bus/pci/devices/${INTERFACE_PCI}/driver/unbind

#Bind the VF to vfio-pci
echo vfio-pci > /sys/bus/pci/devices/${INTERFACE_PCI}/driver_override
echo vfio-pci > /sys/bus/pci/drivers/vfio-pci/bind
```

ADDING VF's TO VIRTIO-FORWARDER

The following changes are made to the Virtio-Forwarder configuration file (**/etc/default/virtioforwarder**) to statically add VF's to the forwarder. The syntax is *<PCI>=<virtio_id>, e.g. 0000:02:08.0=1*

```
#Add VF's 02:08.0 and 02:08.1 statically to the forwarder

VIRTIOFWD_STATIC_VFS=(0000:02:08.0=1 0000:02:08.1=2)
```

VIRTIO-FORWARDER IN VHOSTUSER SERVER MODE

By default, Virtio-Forwarder is configured to run in vhostuser client mode. This configuration is optional but recommended, as it allows reconnection to the VMs when virtio-forwarder is reconfigured and restarted, without having to restart the VMs themselves. Note that client mode requires at least QEMU 2.7.

To run Virtio-Forwarder in vhostuser server mode, make the following changes to the configuration file **/etc/default/virtioforwarder**

```
# Non-blank enables vhostuser client mode
VIRTIOFWD_VHOST_CLIENT=
```

Additionally, the VM must be configured to use vhostuser in client mode. This may be done using the following XML snippet for libvirt

```
<interface type='vhostuser'>
  <mac address='52:54:00:bf:e3:ae'/>
  <source type='unix' path='/tmp/virtio-forwarder/virtio-forwarder1.sock'
mode='client'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x06'
function='0x0'/>
</interface>
```

STARTING VIRTIO-FORWARDER

The Virtio-Forwarder daemon may be started using using systemd.

```
#Start the virtio-forwarder service
systemctl start virtio-forwarder

#Check the status of virtio-forwarder
systemctl status virtio-forwarder
```

AGILIO OPEN VSWITCH CONFIGURATION

An OVS bridge may be created with the VF's to be used added to the bridge to allow the connection between the guests to be confirmed.

A normal rule is added to the bridge to allow it to behave as a learning switch.

```
#Create a bridge named br0
ovs-vsctl add-br br0

#Add VF representors to the bridge
ovs-ofctl add-port br0 eth1 /
-- set Interface eth1 external_ids:virtio_forwarder=1

ovs-ofctl add-port br0 eth2 /
-- set Interface eth2 external_ids:virtio_forwarder=2
#Add NORMAL rule
ovs-ofctl del-flows br0
ovs-ofctl add-flow br0 actions=NORMAL
```

GUEST CONFIGURATION

It is important for the VM memory to be marked as shareable (share=on) and preallocated (prealloc=on and -mem-prealloc), the mem-path must also be correctly specify the hugepage mount point used on the system. The path of the socket must be set to the correct virtio-forwarder vhost-user instance. And the MAC address should be configured as necessary. An example using libvirt:

```
<memoryBacking>
  <hugepages>
    <page size='1048576' unit='KiB' nodeset='0'/>
  </hugepages>
</memoryBacking>

<cpu mode='custom' match='exact'>
  <model fallback='allow'>SandyBridge</model>
  <feature policy='require' name='ssse3'/>
  <numa>
    <cell id='0' cpus='0-1' memory='3670016' unit='KiB'
memAccess='shared'/>
  </numa>
</cpu>
```

The following snippet illustrates how to add a vhost-user interface to the domain:

```
<devices>
  <interface type='vhostuser'>
    <source type='unix' path='/tmp/virtio-forwarder/virtio-forwarder1.sock'
mode='client'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06'
function='0x0'/>
  </interface>
</devices>
```

TESTING CONNECTION

Start Virtio-Forwarder using systemd

```
systemctl start virtio-forwarder
```

Two guests are spawned and one Virtio-Forwarder vhostuser interface is added to each guest respectively. Inside the guests, an IP is configured on the Virtio-Forwarder netdev as follows

```
#Guest_1
ip address add dev ens10 12.0.0.1/24
ip link set dev ens10 up

#Guest_2
ip address add dev ens10 12.0.0.2/24
ip link set dev ens10 up
```

A simple ping may be executed between the two guests to confirm connectivity

```
#From Guest 1
ping 12.0.0.2

64 bytes from 12.0.0.1: icmp_seq=1227 ttl=64 time=0.292 ms
64 bytes from 12.0.0.1: icmp_seq=1228 ttl=64 time=0.216 ms
64 bytes from 12.0.0.1: icmp_seq=1231 ttl=64 time=0.201 ms
```

LIVE MIGRATION

LIVE MIGRATION USING LIBVIRT

Live migration is supported by Virtio-Forwarder and has been tested with QEMU 2.11. The guest configuration must conform to some requirements for live migration. More details on these requirements are listed at http://www.linux-kvm.org/page/Migration#Requirements.

- Apparmor configuration must be correct on both machines.
- VM disk cache must be disabled, e.g. **`<driver name='qemu' type='qcow2' cache='none'/>`** (inside the disk element).
- Same versions of QEMU must be available on both machines.
- Guest storage must be accessible by both source and destination hosts. There are various ways to achieve shared access. For simplicity NFS is used in this example

```
#Install NFS common files and server support on source host
apt-get install nfs-common nfs-kernel-server

#Install NFS common files on destination host
apt-get install nfs-common

#Configure NFS on source host
mkidr /var/lib/libvirt/images/migrate

echo "/var/lib/libvirt/images/migrate/ \
*(rw,no_root_squash)"  > /etc/exports

#restart NFS server on source host
systemctl restart nfs

#Mount shared location on destination host
mkdir -p /var/lib/libvirt/images/migrate/

iptables -F

mount -t nfs <source_host>:/var/lib/libvirt/images/migrate/ \
/var/lib/libvirt/images/migrate/
```

Live migration may be performed with the following command on the source host

```
virsh migrate --live <guest_name> qemu+ssh://<destination_host>/system
```

LIVE MIGRATION IN A ORCHESTRATION SYSTEM

At the time writing, live migration using Virtio-Forwarder within an orchestration system is a work in progress and not yet fully supported.

There are a few basic infrastructure requirements within the orchestration system that is required for live migration:

- Infrastructure to plug guests into the vhostuser path needs to be present. (This would normally be present if the orchestration system supports plugging into the DPDK datapath)

- Infrastructure within the orchestrations system to run DPDK applications should be present
- Extensions to the plug and unplug code needs to be implemented. In the case of Open vSwitch and Openstack these changes need to be made to nova legacy vif

OPEN VSWITCH WITH NORMAL VIRTIO INTERFACES

It is possible to plug libvirt guests into openvSwitch bridges by means of a normal virtIO interface. This option will configure an unaccelerated connection to the guest by bypassing Virtio-Forwarder. Although the openFlow rules will still be offloaded to the smartNIC, the connection to the guest will become the bottleneck as the normal virtIO interface will not be able to cope with the throughput that the smartNIC is capable of.

The following libvirt XML snippet is relevant to create a virtIO interface in the guest

```
   <interface type='bridge'>
      <source bridge='br0'/>
      <virtualport type='openvswitch'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
    </interface>
```