Virtio-forwarder User's Guide

Release 1.1.0-3

Jul 06, 2018

Contents

1 Introduction

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's vhost-user library and services designated VFs by means of the DPDK poll mode driver (PMD) mechanism.

VIO4WD supports up to 64 forwarding instances, where an instance is essentially a VF <-> virtio pairing. Packets received on the VFs are sent on their corresponding virtio backend and vice versa. The relay principle allows a user to benefit from technologies provided by both NICs and the the virtio network driver. A NIC may offload some or all network functions, while virtio enables VM live migration and is also agnostic to the underlying hardware.

2 Requirements

- QEMU version 2.5 (or newer) must be used for the virtual machine hypervisor. The older QEMU 2.3 and 2.4 do work with virtio-forwarder, though there are bugs, less optimised performance and missing features.
- libvirt 1.2.6 or newer (if using libvirt to manage VMs - manually scripted QEMU command line VMs don't require libvirt)
- 2M hugepages must be configured in Linux, a corresponding hugetlbfs mountpoint must exist, and at least 1375 hugepages must be free for use by virtio-forwarder.
- The SR-IOV VFs added to the relay must be bound to the igb_uio driver on the host.

3 Access Control Policies

3.1 libvirt and apparmor

On Ubuntu systems, libvirt's apparmor permissions might need to be modified to allow read/write access to the hugepages directory and library files for QEMU:

```
# in /etc/apparmor.d/abstractions/libvirt-qemu
# for latest QEMU
/usr/lib/x86_64-linux-gnu/qemu/* rmix,
# for access to hugepages
owner "/mnt/huge/libvirt/qemu/**" rw,
owner "/mnt/huge-1G/libvirt/qemu/**" rw,
```

Be sure to substitute the hugetlbfs mountpoints that you use into the above. It may also be prudent to check for any deny lines in the apparmor configuration that may refer to paths used by virtio-forwarder, such as hugepage mounts or vhostuser sockets (default /tmp).

3.2 SELinux

On RHEL or CentOS systems, SELinux's access control policies may need to be to be changed to allow virtio-forwarder to work. The semanage utility can be used to set the svirt_t domain into permissive mode, thereby allowing the functioning of the relay:

```
yum install policycoreutils-python
semanage permissive -a svirt_t
```

4 Hugepages

virtio-forwarder requires 2M hugepages and QEMU/KVM performs better with 1G hugepages. To set up the system for use with libvirt, QEMU and virtio-forwarder, the following should be added to the Linux kernel command line parameters:

```
hugepagesz=2M hugepages=1375 default_hugepagesz=1G hugepagesz=1G
hugepages=8
```

The following could be done after each boot:

```
# Reserve at least 1375 * 2M for virtio-forwarder:
echo 1375 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
# Reserve 8G for application hugepages (modify this as needed):
echo 8 > /sys/kernel/mm/hugepages/hugepages-1048576kB/nr_hugepages
```

Note that reserving hugepages after boot may fail if not enough contiguous free memory is available, and it is therefore recommended to reserve them at boot time with Linux kernel command line parameters. This is especially true for 1G hugepages.

hugetlbfs needs to be mounted on the filesystem to allow applications to create and allocate handles to the mapped memory. The following lines mount the two types of hugepages on /mnt/huge (2M) and /mnt/huge-1G (1G):

```
grep hugetlbfs /proc/mounts | grep -q "pagesize=2M" || \
( mkdir -p /mnt/huge && mount nodev -t hugetlbfs -o rw,pagesize=2M /mnt/huge/ )
grep hugetlbfs /proc/mounts | grep -q "pagesize=1G" || \
( mkdir -p /mnt/huge-1G && mount nodev -t hugetlbfs -o rw,pagesize=1G /mnt/huge-1G/ )
```

Finally, libvirt requires a special directory inside the hugepages mounts with the correct permissions in order to create the necessary per-VM handles:

```
mkdir /mnt/huge-1G/libvirt
mkdir /mnt/huge/libvirt
chown [libvirt-]qemu:kvm -R /mnt/huge-1G/libvirt
chown [libvirt-]qemu:kvm -R /mnt/huge/libvirt
```

Note: After these mounts have been prepared, the libvirt daemon will probably need to be restarted.

---

5 Installation

virtio-forwarder packages are hosted on copr and ppa. To install, add the applicable repository and launch the appropriate package manager:

```
# rpms
yum install yum-plugin-copr
yum copr enable netronome/virtio-forwarder
yum install virtio-forwarder

# debs
add-apt-repository ppa:netronome/virtio-forwarder
apt-get update
apt-get install virtio-forwarder
```

The package install configures virtio-forwarder as a systemd/upstart service. Boot time startup can be configured using the appropriate initialization utility, e.g. `systemctl enable virtio-forwarder`.

After installation, the software can be manually started using the following command:

```
systemctl start virtio-forwarder # systemd
start virtio-forwarder # upstart
```

Configuration variables taken into account at startup can be set in the `/etc/default/virtioforwarder` file. The next section highlights some important options.

The *virtio-forwarder* daemon can be stopped by substituting `stop` in the start commands of the respective initialization utilities.

An additional CPU load balancing component is installed alongside virtio-forwarder. The service, *vio4wd_core_scheduler*, is managed exactly like virtio-forwarder with regard to starting, stopping and configuration.

6 Daemon Configuration

Both the virtio-forwarder and vio4wd_core_scheduler daemons read from `/etc/default/virtioforwarder` at startup. The file takes the form of `variable=value` entries, one per line. Lines starting with the "#" character are treated as comments and ignored. The file comes pre-populated with sane default values, but may require alterations to comply with different setups. The following table lists a subset of the available options and their use:

Table 2: virtio-forwarder Configuration Variables

Name / Description	Valid values	Default
VIRTIOFWD_CPU_MASK CPUs to use for worker threads: either comma separated integers or, hex bitmap starting with 0x.	0 - number of host CPU	1,2
VIRTIOFWD_LOG_LEVEL Log threshold 0-7 (least to most verbose).	0-7	6
VIRTIOFWD_OVSDB_SOCK_PATH Path to the ovsdb socket file used for port control.	System path	/usr/local/var/run/openvswitch/db.sock
VIRTIOFWD_HUGETLBFS_MOUNT_POINT Mount path to hugepages for vhost-user communication with VMs. This must match the path configured for libvirt/QEMU.	System path	/mnt/huge
VIRTIOFWD_SOCKET_OWNER vhost-user unix socket ownership username.	Username	libvirt-qemu

VIRTIOFWD_SOCKET_GROUP vhost-user unix socket ownership groupname.	Groupname	kvm
VIO4WD_CORE_SCHED_ENABLE Use dynamic CPU load balancing. Toggle flag to enable the CPU migration API to be exposed. vio4wd_core_scheduler requires this option to function.	true or false	false
VIRTIOFWD_CPU_PINS Relay CPU pinnings. A semicolon-delimited list of strings specifying which CPU(s) to use for the specified relay instances.	<vf>:<cpu>[,<cpu>]	None
VIRTIOFWD_DYNAMIC_SOCKETS Enable dynamic sockets. virtio-forwarder will not create or listen to any sockets when dynamic sockets are enabled. Instead, socket registration/deregistration must ensue through the ZMQ port control client.	true or false	false

7 Adding VF Ports to Virtio-forwarder

virtio-forwarder implements different methods for the addition and removal of VFs and bonds. Depending on the use case, one of the following may be appropriate:

- **ZeroMQ port control** for the purpose of manual device and socket management at run-time. Run `/usr/libexec/virtio-forwarder/virtioforwarder_port_control_tester.py -h` for usage guidelines. To enable ZeroMQ VF management, set `VIRTIOFWD_ZMQ_PORT_CONTROL_EP` to an appropriate path in the configuration file.

  The port control client is the preferred device management tool, and is the only utility that can exercise all the device related features of virtio-forwarder. Particularly, bond creation/deletion, and dynamic socket registration/deregistration are only exposed to the port control client. The examples below demonstrate the different modes of operation:

  - **Add VF**

    ```
    virtioforwarder_port_control_tester.py add --virtio-id=<ID> \
    --pci-addr=<PCI_ADDR>
    ```

  - **Remove VF**

    ```
    virtioforwarder_port_control_tester.py remove --virtio-id=<ID> \
    --pci-addr=<PCI_ADDR>
    ```

  - **Add bond**

```
virtioforwarder_port_control_tester.py add --virtio-id=<ID> \
--name=<BOND_NAME> --pci-addr=<PCI_ADDR> --pci-addr=<PCI_ADDR> \
[--mode=<MODE>]
```

  - **Remove bond**

```
virtioforwarder_port_control_tester.py remove --virtio-id=<ID> \
--name=<BOND_NAME>
```

  - **Add device <-> vhost-user socket pair**

```
virtioforwarder_port_control_tester.py add_sock \
--vhost-path=</path/to/vhostuser.sock> --pci-addr=<PCI_ADDR> \
[--pci-addr=<PCI_ADDR> --name=<BOND_NAME> [--mode=<MODE>]]
```

  - **Remove device <-> vhost-user socket pair**

```
virtioforwarder_port_control_tester.py remove_sock \
--vhost-path=</path/to/vhostuser.sock> \
(--pci-addr=<PCI_ADDR>|--name=<BOND_NAME>)
```

Note:

- A bond operation is assumed when multiple PCI addresses are provided.
- Bond names are required to start with *net_bonding*.
- Socket operations only apply if virtio-forwarder was started with the `VIRTIOFWD_DYNAMIC_SOCKETS` option enabled.

- **Static VF entries** in /etc/default/virtioforwarder. VFs specified here are added when the daemon starts. The `VIRTIOFWD_STATIC_VFS` variable is used for this purpose, with the syntax *<PCI>=<virtio_id>*, e.g. *0000:05:08.1=1*. Multiple entries can be specified using bash arrays. The following examples are all valid:
  - VIRTIOFWD_STATIC_VFS=0000:05:08.1=1
  - VIRTIOFWD_STATIC_VFS=(0000:05:08.1=1)
  - VIRTIOFWD_STATIC_VFS=(0000:05:08.1=1 0000:05:08.2=2 0000:05:08.3=3)
- **OVSDB monitor:** The ovs-vsctl command manipulates the OVSDB, which is monitored for changes by virtio-forwarder. To add a VF to the virtio-forwarder, the ovs-vsctl command can be used with a special external_ids value containing an indication to use the relay. The bridge name br-virtio in this example is arbitrary, any bridge name may be used:

```
ovs-vsctl add-port br-virtio eth100 -- set interface \
eth100 external_ids:virtio_forwarder=1
```

  Note that the ports in the OVSDB remain configured across OvS restarts, and when virtio-forwarder starts it will find the initial list of ports with associated virtio-forwarder indications and recreate the necessary associations.

  Changing an interface with no virtio-forwarder indication to one with a virtio- forwarder indication, or changing one with a virtio-forwarder indication to one without a virtio-forwarder indication also works. e.g.

```
# add to OvS bridge without virtio-forwarder (ignored by virtio-forwarder)
ovs-vsctl add-port br-virtio eth100
# add virtio-forwarder (detected by virtio-forwarder)
```

(continues on next page)

(continued from previous page)

```
ovs-vsctl set interface eth100 external_ids:virtio_forwarder=1
# remove virtio-forwarder (detected by virtio-forwarder and removed from
# relay, but remains on OvS bridge)
ovs-vsctl remove interface eth100 external_ids virtio_forwarder
```

The externals_ids of a particular interface can be viewed with ovs-vsctl as follows:

```
ovs-vsctl list interface eth100 | grep external_ids
```

A list of all the interfaces with external_ids can be queried from OVSDB:

```
ovsdb-client --pretty -f list dump Interface name external_ids | \
grep -A2 -E "external_ids.*: {.+}"
```

- **Inter-process communication (IPC)** which implements a file monitor for VF management. Set `VIRTIOFWD_IPC_PORT_CONTROL` in the configuration file to non-null to enable.

Note: ZMQ, OVSDB and IPC port control are mutually exclusive.

Warning: Relayed VFs cannot be used for SR-IOV passthrough while in use by virtio- forwarder, as libvirt will disregard the igb_uio binding of relayed VFs when establishing a passthrough connection. This causes irrevocable interference with the igb_uio module, leading to an eventual segmentation fault.

8 CPU Affinities

The `VIRTIOFWD_CPU_PINS` variable in the configuration file can be used to control VF relay CPU affinities. The format of the option is `--virtio-cpu=<vf>:<cpu>[,<cpu>]`, where `<cpu>` must be a valid CPU enabled in the `VIRTIOFWD_CPU_MASK` configuration option. Specifying two CPUs for a particular VF allows the VF-to-virtio and virtio-to-VF relay directions to be serviced by separate CPUs, enabling higher performance to a particular virtio endpoint in a VM. If a given VF is not bound to a CPU (or CPUs), then that VF relay will be assigned to the least busy CPU in the list of CPUs provided in the configuration. The option may contain multiple affinity specifiers, one for each VF number.

9 CPU Load Balancing

In some scenarios, virtio-forwarder's CPU assignments may result in poor relay to CPU affinities due to the network load being unevenly distributed among worker cores. A relay's throughput will suffer when it is serviced by worker cores under excessive processing load. Manual pinnings may also prove suboptimal under varying network requirements. The external vio4wd_core_scheduler load balancing daemon is included to address this issue. The balancer daemon gathers network load periodically in order to determine and apply an optimal affinity solution. ZeroMQ is used for inter-process communication. Note that VIO4WD_CORE_SCHED_ENABLE must be explicitely set to true for virtio-forwarder to create and listen on the ZeroMQ endpoint required for CPU migration.

Note: When running, the load balancer may overwrite manual pinnigs at any time!

10 Running Virtual Machines

QEMU virtual machines can be run manually on the command line, or by using libvirt to manage them. To use QEMU manually with the vhost-user backed VirtIO which the virtio-forwarder provides, the following example can be used:

```
-object memory-backend-file,id=mem,size=3584M,mem-path=/mnt/huge-1G,share=on,
↪prealloc=on \
-numa node,memdev=mem -mem-prealloc \
-chardev socket,id=chr0,path=/tmp/virtio-forwarder1.sock \
-netdev type=vhost-user,id=guest3,chardev=chr0,vhostforce \
-device virtio-net-pci,netdev=guest3,csum=off,gso=off,guest_tso4=off,guest_tso6=off,\
guest_ecn=off,mac=00:03:02:03:04:01
```

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 specified to 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 may be configured as needed.

Virtual machines may also be managed using libvirt, and this requires some specific XML snippets in the libvirt VM domain specification file:

```
<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>
```

If only 2M hugepages are in use on the system, the domain can be configured with the following page size:

```
<page size='2' unit='MiB' nodeset='0'/>
```

Note, the emulated CPU requires SSSE3 instructions for DPDK support.

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

```
<devices>
  <interface type='vhostuser'>
    <source type='unix' path='/tmp/virtio-forwarderRELAYID.sock' mode='client'/>
    <model type='virtio'/>
    <alias name='net1'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
  </interface>
</devices>
```

Note: When starting the domain, make sure that the permissions are correctly set on the relay vhost-user socket, as well as adding the required permissions to the apparmor profile. The `VIRTIOFWD_SOCKET_OWNER` and `VIRTIOFWD_SOCKET_GROUP` options in the configuration file can also be used to set the permissions on the vhost-user sockets.

11 Using vhost-user Client Mode

The `VIRTIOFWD_VHOST_CLIENT` option can be used to put virtio-forwarder in vhostuser client mode instead of the default server mode. This requires the VM to use QEMU v2.7 or newer, and the VM must be configured to use vhostuser server mode, e.g. for libvirt:

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

or when using a QEMU cmdline directly:

```
-chardev socket,id=charnet1,path=/tmp/virtio-forwarder1.sock,server
```

The advantage of this is that virtio-forwarder will attempt to re-establish broken vhostuser connections automatically. In particular, this allows virtio-forwarder to be restarted while a VM is running (and still have virtio connectivity afterwards), as well as have a VM be restarted while virtio-forwarder is running. In the default virtio-forwarder vhostuser server mode, only the latter is possible.

12 Multiqueue Virtio

virtio-forwarder supports multiqueue virtio up to a maximum of 32 queues, where the QEMU VM is configured in the standard way. For libvirt configured VMs, libvirt version >= 1.2.17 is required for multiqueue support, and then one can simply add `<driver queues='4'/>` inside the vhostuser interface chunk in libvirt XML, where 4 is the number of queues required, e.g.:

```
<interface type='vhostuser'>
  <mac address='52:54:00:bf:e3:ae'/>
  <source type='unix' path='/tmp/virtio-forwarder1.sock' mode='client'/>
  <model type='virtio'/>
  <driver queues='4'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
</interface>
```

This results in the following cmdline params to QEMU:

```
-chardev socket,id=charnet1,path=/tmp/virtio-forwarder1.sock -netdev type=vhost-user,\
id=hostnet1,chardev=charnet1,queues=4 -device virtio-net-pci,mq=on,vectors=10,\
netdev=hostnet1,id=net1,mac=52:54:00:bf:e3:ae,bus=pci.0,addr=0x6
```

(i.e. the queues item in netdev option, and the mq and vectors items in device option, where the vectors value must be (queues+1)*2)

To enable the multiqueue inside the VM:

```
# to see max and current queues:
ethtool -l eth1
# to set queues
ethtool -L eth1 combined 4
```

13 Performance Tuning

Important aspects that influence performance are resource contention, and CPU and memory NUMA affinities. The following are general guidelines to follow for a performance oriented setup:

- Pin VM VCPUs.
- Dedicate worker CPUs for relays.
- Do not make any overlapping CPU assignments.
- Set the NUMA affinity of a VM's backing memory and ensure that it matches the VCPUs. The numatune libvirt xml snippet can be used for this.
- Keep hyperthread partners idle.
- Disable interrupts on the applicable CPUs.
- Keep all components on the same NUMA. If you want to utilize the other NUMA, assign everything (VCPUs, VM memory, VIO4WD workers) to that NUMA so that only the PCI device is cross-socket.

If a VM's backing memory is confined to a particular NUMA, virtio-forwarder will automatically align the corresponding relay's memory pool with the VM's upon connection in order to limit QPI crossings. Moreover, the CPU load balancing daemon will only consider CPUs that are local to a relay's NUMA to service it.

14 Debugging Utilities

Helper and debugging scripts are located in /usr/libexec/virtio-forwarder/. Here are pointers to using some of the more useful ones:

- virtioforwarder_stats.py: Gathers statistics (including rate stats) from running relay instances.
- core_pinner.py: Manually pin relay instances to CPUs at runtime. Uses the same syntax as the environment file, that is, –virtio-cpu=R_N:C_i,C_j. Run without arguments to get the current relay to CPU mapping. Note that the mappings may be overridden by the load balancer if it is also running. The same is true for mappings provided in the configuration file.
- monitor_load.py: Provides a bar-like representation of the current load on worker CPUs. Useful to monitor the work of the load balancer.

System logs can be viewed by running `journalctl -u virtio-forwarder -u vio4wd_core_scheduler` on systemd-enabled systems. Syslog provides the same information on older systems.

15 Using VirtIO 1.0

To enable VirtIO 1.0 (as opposed to legacy VirtIO), the backend virtual PCI device provided by QEMU needs to be enabled. Using QEMU 2.5, you need to supply an extra cmdline parameter to prevent VirtIO 1.0 support from being disabled (it is disabled by default, since there are apparently still known issues with performance, stability and live migration):

```
-global virtio-pci.disable_modern=off
```

This can be done in a libvirt domain by ensuring the domain spec starts with something like:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
```

and just prior to the closing </domain> tag adding the following:

```
<qemu:commandline>
  <qemu:arg value='-global'/>
  <qemu:arg value='virtio-pci.disable-modern=off'/>
</qemu:commandline>
```

In addition to this, the vhost or vhost-user connected to the device in QEMU must support VirtIO 1.0. The vhostuser interface which virtio-forwarder supplies does support this, but if the host is running a Linux kernel older than 4.0, you likely won't have vhost-net (kernel) support for any network interfaces in your QEMU VM which are not connected to virtio-forwarder, for example if you have a bridged management network interface. Libvirt will by default use vhost net for that, you can disable vhost-net by adding <driver name='qemu'/> to the relevant bridge interface as follows:

```
<interface type='bridge'>
  ...
  <model type='virtio'/>
  <driver name='qemu'/>
  ...
</interface>
```

To use VirtIO 1.0 with DPDK inside a VM, you will need to use DPDK 16.04. To use a VirtIO 1.0 netdev in the VM, the VM must be running Linux kernel version 4.0 or newer.

16 VM Live Migrate with libvirt

The virtio-forwarder is compatible with QEMU VM live migration as abstracted by libvirt, and has been tested using QEMU 2.5 with libvirt 1.2.16. The VM configuration must conform to some requirements to allow live migration to take place. In short:

- VM disk image must be accessible over shared network storage accessible to the source and destination machines.
- Same versions of QEMU must be available on both machines.
- 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).
- The hugepages for both machines must be correctly configured.
- Ensure both machines have Linux kernels new enough to support vhost-net live migration for any virtio network devices not using the vhostuser interface, or configure such interfaces to only use vanilla QEMU virtio backend support, e.g. `<model type='virtio'/> <driver name='qemu'/>` (inside the relevant interface elements).

The VM live migration can be initiated from the source machine by giving the VM name and target user&hostname as follows:

```
virsh migrate --live <vm_name> qemu+ssh://<user@host>/system
```

The `--verbose` argument can optionally be added for extra information. If all goes well, virsh list on the source machine should no longer show <vm_name> and instead it should appear in the output of virsh list on the destination machine. If anything goes wrong, the following log files often have additional details to help troubleshoot the problem:

```
journalctl
/var/log/syslog
/var/log/libvirt/libvirt.log
/var/log/libvirt/qemu/<vm_name>.log
```

In the simplest scenario, the source and destination machines have the same VM configuration, particularly with respect to the vhostuser socket used on virtio- forwarder. It may be handy to configure the vhostuser socket in the VM to point to a symlink file which links to one of the virtio-forwarder sockets. This is one way to allow the source and destination machines to use different vhostuser sockets if necessary. For example, on the source machine one might be using a symlink called /tmp/vm_abc.sock linking to /tmp/virtio-forwarder1.sock, while on the destination machine /tmp/vm_abc.sock might link to /tmp/virtio-forwarder13.sock.

It is also possible to migrate between machines where one is using virtio-forwarder, and the other is using a different virtio backend driver (could be a different vhostuser implementation, or could even be vhost-net or plain QEMU backend). The key to achieving this is the *–xml* parameter for the virsh migrate command (virsh help migrate reveals: `--xml <string> filename containing updated XML for the target`).

Here is an example of the procedure to migrate from a vhostuser VM (connected to virtio-forwarder) to a nonvhostuser VM:

On the destination machine, set up a libvirt network that you want to migrate the interface onto, e.g. named 'migrate', by passing the following XML file to virsh net-define <xml_file> and running it with virsh net-start migrate; virsh net-autostart migrate:

```
<network>
  <name>migrate</name>
  <bridge name='migratebr0' stp='off' delay='0'/>
</network>
```

On the source machine (where the VM is defined to use vhostuser connected to virtio-forwarder), dump the VM XML to a file by running `virsh dumpxml <vm_name> >domain.xml`. Edit the domain.xml file to change the vhostuser interfaces to be sourced by the migrate network, i.e. change these:

```
<interface type='vhostuser'>
  <mac address='00:0a:00:00:00:00'/>
  <source type='unix' path='/tmp/virtio-forwarder0.sock' mode='client'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
</interface>
```

to these:

```
<interface type='network'>
  <mac address='00:0a:00:00:00:00'/>
  <source network='migrate'>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
</interface>
```

Finally, once you have this modified domain.xml file, the VM can be migrated as follows:

```
virsh migrate --live <vm_name> qemu+ssh://<user@host>/system --xml domain.xml
```

Migrating from a non virtio-forwarder machine to a virtio-forwarder machine follows this same procedure in reverse; a new XML file is made where the migrate network interfaces are changed to vhostuser interfaces.