APPENDIX A

Libvirt XML Schemas

This appendix covers the XML schemas used by libvirt. Each major section in the appendix describes a single libvirt domain. Everything needed to completely describe all the elements of a libvirt domain are contained in the schema.

The Domain Schema

The Domain schema completely describes a libvirt domain, i.e., a virtual machine. The domain can be any of the supported libvirt types including QEMU/KVM, XEN, and so on. Obviously, some of the schema sections apply to only some of the supported domain types, so it makes sense that you need to include only the parts of the schema that apply to your domain. The same applies to devices and features. If your domain does not support or have certain devices or features, then simply do not include those sections.

The Domain XML schema consists of a large number of XML elements (Listing A-1). The documentation for this schema is quite extensive, but in most cases the uses of the elements in the schema are obvious. If you need documentation, refer to `https://libvirt.org/formatdomain.html` for more specific information about each element and attribute.

W. D. Ashley, *Foundations of Libvirt Development*,
https://doi.org/10.1007/978-1-4842-4862-1

Listing A-1. The Domain Schema

```
<!--General metadata -->

<domain type='kvm' id='1'>
  <name>MyGuest</name>
  <uuid>4dea22b3-1d52-d8f3-2516-782e98ab3fa0</uuid>
  <genid>43dc0cf8-809b-4adb-9bea-a9abb5f3d90e</genid>
  <title>A short description - title - of the domain</title>
  <description>Some human readable description</description>
  <metadata>
    <app1:foo xmlns:app1="http://app1.org/app1/">..</app1:foo>
    <app2:bar xmlns:app2="http://app1.org/app2/">..</app2:bar>
  </metadata>
  <bootloader>/usr/bin/pygrub</bootloader>
  <bootloader_args>--append single</bootloader_args>

  <!-- BIOS bootloader -->

  <os>
    <type>hvm</type>
    <loader readonly='yes' secure='no' type='rom'>
        /usr/lib/xen/boot/hvmloader</loader>
    <nvram template='/usr/share/OVMF/OVMF_VARS.fd'>
        /var/lib/libvirt/nvram/guest_VARS.fd</nvram>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
    <bootmenu enable='yes' timeout='3000'/>
    <smbios mode='sysinfo'/>
    <bios useserial='yes' rebootTimeout='0'/>
  </os>
```

```
<!-- Host bootloader -->

<bootloader>/usr/bin/pygrub</bootloader>
<bootloader_args>--append single</bootloader_args>

<!-- Direct kernel boot -->

<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <kernel>/root/f8-i386-vmlinuz</kernel>
  <initrd>/root/f8-i386-initrd</initrd>
  <cmdline>console=ttyS0 ks=http://example.com/f8-i386/os/
  </cmdline>
  <dtb>/root/ppc.dtb</dtb>
  <acpi>
    <table type='slic'>/path/to/slic.dat</table>
  </acpi>
</os>

<!-- Container boot -->

<os>
  <type arch='x86_64'>exe</type>
  <init>/bin/systemd</init>
  <initarg>--unit</initarg>
  <initarg>emergency.service</initarg>
  <initenv name='MYENV'>some value</initenv>
  <initdir>/my/custom/cwd</initdir>
  <inituser>tester</inituser>
  <initgroup>1000</initgroup>
</os>
```

```
<idmap>
  <uid start='0' target='1000' count='10'/>
  <gid start='0' target='1000' count='10'/>
</idmap>

<!-- SMBIOS System Information -->

<os>
<smbios mode='sysinfo'/>
</os>
<sysinfo type='smbios'>
  <bios>
    <entry name='vendor'>LENOVO</entry>
  </bios>
  <system>
    <entry name='manufacturer'>Fedora</entry>
    <entry name='product'>Virt-Manager</entry>
    <entry name='version'>0.9.4</entry>
  </system>
  <baseBoard>
    <entry name='manufacturer'>LENOVO</entry>
    <entry name='product'>20BE0061MC</entry>
    <entry name='version'>0B98401 Pro</entry>
    <entry name='serial'>W1KS427111E</entry>
  </baseBoard>
  <chassis>
    <entry name='manufacturer'>Dell Inc.</entry>
    <entry name='version'>2.12</entry>
    <entry name='serial'>65X0XF2</entry>
    <entry name='asset'>40000101</entry>
    <entry name='sku'>Type3Sku1</entry>
  </chassis>
```

```
    <oemStrings>
      <entry>myappname:some arbitrary data</entry>
      <entry>otherappname:more arbitrary data</entry>
    </oemStrings>
  </sysinfo>

  <!-- CPU Allocation -->

  <vcpu placement='static' cpuset="1-4,^3,6" current="1">2</vcpu>
  <vcpus>
    <vcpu id='0' enabled='yes' hotpluggable='no' order='1'/>
    <vcpu id='1' enabled='no' hotpluggable='yes'/>
  </vcpus>

  <!-- IOThreads Allocation -->

  <iothreads>4</iothreads>
  <iothreadids>
    <iothread id="2"/>
    <iothread id="4"/>
    <iothread id="6"/>
    <iothread id="8"/>
  </iothreadids>

 <!-- CPU Tuning -->

  <cputune>
    <vcpupin vcpu="0" cpuset="1-4,^2"/>
    <vcpupin vcpu="1" cpuset="0,1"/>
    <vcpupin vcpu="2" cpuset="2,3"/>
    <vcpupin vcpu="3" cpuset="0,4"/>
    <emulatorpin cpuset="1-3"/>
    <iothreadpin iothread="1" cpuset="5,6"/>
    <iothreadpin iothread="2" cpuset="7,8"/>
    <shares>2048</shares>
    <period>1000000</period>
    <quota>-1</quota>
    <global_period>1000000</global_period>
    <global_quota>-1</global_quota>
    <emulator_period>1000000</emulator_period>
    <emulator_quota>-1</emulator_quota>
    <iothread_period>1000000</iothread_period>
    <iothread_quota>-1</iothread_quota>
    <vcpusched vcpus='0-4,^3' scheduler='fifo' priority='1'/>
    <iothreadsched iothreads='2' scheduler='batch'/>
    <cachetune vcpus='0-3'>
      <cache id='0' level='3' type='both' size='3' unit='MiB'/>
      <cache id='1' level='3' type='both' size='3' unit='MiB'/>
      <monitor level='3' vcpus='1'/>
      <monitor level='3' vcpus='0-3'/>
    </cachetune>
    <cachetune vcpus='4-5'>
      <monitor level='3' vcpus='4'/>
      <monitor level='3' vcpus='5'/>
    </cachetune>
    <memorytune vcpus='0-3'>
      <node id='0' bandwidth='60'/>
    </memorytune>
  </cputune>

  <!-- Memory Allocation -->

  <maxMemory slots='16' unit='KiB'>1524288</maxMemory>
  <memory unit='KiB'>524288</memory>
  <currentMemory unit='KiB'>524288</currentMemory>
```

```
<!-- Memory Backing -->

<memoryBacking>
  <hugepages>
    <page size="1" unit="G" nodeset="0-3,5"/>
    <page size="2" unit="M" nodeset="4"/>
  </hugepages>
  <nosharepages/>
  <source type="file|anonymous|memfd"/>
  <access mode="shared|private"/>
  <allocation mode="immediate|ondemand"/>
</memoryBacking>

<!-- Memory Tuning -->

<memtune>
  <hard_limit unit='G'>1</hard_limit>
  <soft_limit unit='M'>128</soft_limit>
  <swap_hard_limit unit='G'>2</swap_hard_limit>
  <min_guarantee unit='bytes'>67108864</min_guarantee>
</memtune>

<!-- NUMA Node Tuning -->

<numatune>
  <memory mode="strict" nodeset="1-4,^3"/>
  <memnode cellid="0" mode="strict" nodeset="1"/>
  <memnode cellid="2" mode="preferred" nodeset="2"/>
</numatune>

<!-- Block I/O Tuning -->

<blkiotune>
  <weight>800</weight>
  <device>
```

```
      <path>/dev/sda</path>
      <weight>1000</weight>
    </device>
    <device>
      <path>/dev/sdb</path>
      <weight>500</weight>
      <read_bytes_sec>10000</read_bytes_sec>
      <write_bytes_sec>10000</write_bytes_sec>
      <read_iops_sec>20000</read_iops_sec>
      <write_iops_sec>20000</write_iops_sec>
    </device>
  </blkiotune>

  <!-- Resource partitioning -->

  <resource>
    <partition>/virtualmachines/production</partition>
  </resource>

  <!-- CPU model and topology -->

  <cpu match='exact'>
    <model fallback='allow'>core2duo</model>
    <vendor>Intel</vendor>
    <topology sockets='1' cores='2' threads='1'/>
    <cache level='3' mode='emulate'/>
    <feature policy='disable' name='lahf_lm'/>
  </cpu>
  <cpu mode='host-model'>
    <model fallback='forbid'/>
    <topology sockets='1' cores='2' threads='1'/>
  </cpu>
```

```
<cpu mode='host-passthrough'>
  <cache mode='passthrough'/>
  <feature policy='disable' name='lahf_lm'/>
</cpu>
<cpu>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>
<cpu>
  <numa>
    <cell id='0' cpus='0-3' memory='512000' unit='KiB'
    discard='yes'/>
    <cell id='1' cpus='4-7' memory='512000' unit='KiB'
    memAccess='shared'/>
  </numa>
  <numa>
    <cell id='0' cpus='0,4-7' memory='512000' unit='KiB'>
      <distances>
        <sibling id='0' value='10'/>
        <sibling id='1' value='21'/>
        <sibling id='2' value='31'/>
        <sibling id='3' value='41'/>
      </distances>
    </cell>
    <cell id='1' cpus='1,8-10,12-15' memory='512000' unit='KiB'
        memAccess='shared'>
      <distances>
        <sibling id='0' value='21'/>
        <sibling id='1' value='10'/>
        <sibling id='2' value='21'/>
        <sibling id='3' value='31'/>
      </distances>
    </cell>
```

```
      <cell id='2' cpus='2,11' memory='512000' unit='KiB'
      memAccess='shared'>
        <distances>
          <sibling id='0' value='31'/>
          <sibling id='1' value='21'/>
          <sibling id='2' value='10'/>
          <sibling id='3' value='21'/>
        </distances>
      </cell>
      <cell id='3' cpus='3' memory='512000' unit='KiB'>
        <distances>
          <sibling id='0' value='41'/>
          <sibling id='1' value='31'/>
          <sibling id='2' value='21'/>
          <sibling id='3' value='10'/>
        </distances>
      </cell>
    </numa>
  </cpu>

  <!-- Events configuration -->

  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <on_lockfailure>poweroff</on_lockfailure>

  <!-- Power Management -->

  <pm>
    <suspend-to-disk enabled='no'/>
    <suspend-to-mem enabled='yes'/>
  </pm>
```

```
<!-- Hypervisor features -->

<features>
  <pae/>
  <acpi/>
  <apic/>
  <hap/>
  <privnet/>
  <hyperv>
    <relaxed state='on'/>
    <vapic state='on'/>
    <spinlocks state='on' retries='4096'/>
    <vpindex state='on'/>
    <runtime state='on'/>
    <synic state='on'/>
    <reset state='on'/>
    <vendor_id state='on' value='KVM Hv'/>
    <frequencies state='on'/>
    <reenlightenment state='on'/>
    <tlbflush state='on'/>
    <ipi state='on'/>
    <evmcs state='on'/>
  </hyperv>
  <kvm>
    <hidden state='on'/>
  </kvm>
  <pvspinlock state='on'/>
  <gic version='2'/>
  <ioapic driver='qemu'/>
  <hpt resizing='required'>
    <maxpagesize unit='MiB'>16</maxpagesize>
  </hpt>
```

```
    <vmcoreinfo state='on'/>
    <smm state='on'>
      <tseg unit='MiB'>48</tseg>
    </smm>
    <htm state='on'/>
  </features>

  <!-- Time keeping -->

  <clock offset='localtime'>
    <timer name='rtc' tickpolicy='catchup' track='guest'>
      <catchup threshold='123' slew='120' limit='10000'/>
    </timer>
    <timer name='pit' tickpolicy='delay'/>
  </clock>

  <!-- Performance monitoring events -->

  <perf>
    <event name='cmt' enabled='yes'/>
    <event name='mbmt' enabled='no'/>
    <event name='mbml' enabled='yes'/>
    <event name='cpu_cycles' enabled='no'/>
    <event name='instructions' enabled='yes'/>
    <event name='cache_references' enabled='no'/>
    <event name='cache_misses' enabled='no'/>
    <event name='branch_instructions' enabled='no'/>
    <event name='branch_misses' enabled='no'/>
    <event name='bus_cycles' enabled='no'/>
    <event name='stalled_cycles_frontend' enabled='no'/>
    <event name='stalled_cycles_backend' enabled='no'/>
    <event name='ref_cpu_cycles' enabled='no'/>
    <event name='cpu_clock' enabled='no'/>
    <event name='task_clock' enabled='no'/>
    <event name='page_faults' enabled='no'/>
    <event name='context_switches' enabled='no'/>
    <event name='cpu_migrations' enabled='no'/>
    <event name='page_faults_min' enabled='no'/>
    <event name='page_faults_maj' enabled='no'/>
    <event name='alignment_faults' enabled='no'/>
    <event name='emulation_faults' enabled='no'/>
  </perf>

  <!-- Devices -->

  <devices>
    <emulator>/usr/lib/xen/bin/qemu-dm</emulator>
  </devices>
  <devices>
    <disk type='file'>
      <alias name='ua-myDisk'/>
    </disk>
    <interface type='network' trustGuestRxFilters='yes'>
      <alias name='ua-myNIC'/>
    </interface>
    ...
  </devices>

  <!-- Hard drives, floppy disks, CDROMs -->

  <devices>
    <disk type='file' snapshot='external'>
      <driver name="tap" type="aio" cache="default"/>
      <source file='/var/lib/xen/images/fv0' startupPolicy=
      'optional'>
        <seclabel relabel='no'/>
      </source>
```

```
      <target dev='hda' bus='ide'/>
      <iotune>
        <total_bytes_sec>10000000</total_bytes_sec>
        <read_iops_sec>400000</read_iops_sec>
        <write_iops_sec>100000</write_iops_sec>
      </iotune>
      <boot order='2'/>
      <encryption type='...'>
        ...
      </encryption>
      <shareable/>
      <serial>
        ...
      </serial>
    </disk>
    <disk type='network'>
      <driver name="qemu" type="raw" io="threads" ioeventfd="on"
          event_idx="off"/>
      <source protocol="sheepdog" name="image_name">
        <host name="hostname" port="7000"/>
      </source>
      <target dev="hdb" bus="ide"/>
      <boot order='1'/>
      <transient/>
      <address type='drive' controller='0' bus='1' unit='0'/>
    </disk>
    <disk type='network'>
      <driver name="qemu" type="raw"/>
      <source protocol="rbd" name="image_name2">
        <host name="hostname" port="7000"/>
        <snapshot name="snapname"/>
        <config file="/path/to/file"/>
        <auth username='myuser'>
          <secret type='ceph' usage='mypassid'/>
        </auth>
      </source>
      <target dev="hdc" bus="ide"/>
    </disk>
    <disk type='block' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <target dev='hdd' bus='ide' tray='open'/>
      <readonly/>
    </disk>
    <disk type='network' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <source protocol="http" name="url_path">
        <host name="hostname" port="80"/>
      </source>
      <target dev='hde' bus='ide' tray='open'/>
      <readonly/>
    </disk>
    <disk type='network' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <source protocol="https" name="url_path">
        <host name="hostname" port="443"/>
      </source>
      <target dev='hdf' bus='ide' tray='open'/>
      <readonly/>
    </disk>
    <disk type='network' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <source protocol="ftp" name="url_path">
        <host name="hostname" port="21"/>
      </source>
```

```
      <target dev='hdg' bus='ide' tray='open'/>
      <readonly/>
    </disk>
    <disk type='network' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <source protocol="ftps" name="url_path">
        <host name="hostname" port="990"/>
      </source>
      <target dev='hdh' bus='ide' tray='open'/>
      <readonly/>
    </disk>
    <disk type='network' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <source protocol="tftp" name="url_path">
        <host name="hostname" port="69"/>
      </source>
      <target dev='hdi' bus='ide' tray='open'/>
      <readonly/>
    </disk>
    <disk type='block' device='lun'>
      <driver name='qemu' type='raw'/>
      <source dev='/dev/sda'>
        <reservations managed='no'>
          <source type='unix' path='/path/to/qemu-pr-helper'
          mode='client'/>
        </reservations>
      <target dev='sda' bus='scsi'/>
      <address type='drive' controller='0' bus='0' target='3'
      unit='0'/>
    </disk>
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw'/>
      <source dev='/dev/sda'/>
      <geometry cyls='16383' heads='16' secs='63' trans='lba'/>
      <blockio logical_block_size='512' physical_block_size='4096'/>
      <target dev='hdj' bus='ide'/>
    </disk>
    <disk type='volume' device='disk'>
      <driver name='qemu' type='raw'/>
      <source pool='blk-pool0' volume='blk-pool0-vol0'/>
      <target dev='hdk' bus='ide'/>
    </disk>
    <disk type='network' device='disk'>
      <driver name='qemu' type='raw'/>
      <source protocol='iscsi' name='iqn.2013-07.com.
      example:iscsi-nopool/2'>
        <host name='example.com' port='3260'/>
        <auth username='myuser'>
          <secret type='iscsi' usage='libvirtiscsi'/>
        </auth>
      </source>
      <target dev='vda' bus='virtio'/>
    </disk>
    <disk type='network' device='lun'>
      <driver name='qemu' type='raw'/>
      <source protocol='iscsi' name='iqn.2013-07.com.
      example:iscsi-nopool/1'>
        <host name='example.com' port='3260'/>
        <auth username='myuser'>
          <secret type='iscsi' usage='libvirtiscsi'/>
        </auth>
      </source>
      <target dev='sdb' bus='scsi'/>
    </disk>
```

```
    <disk type='network' device='lun'>
      <driver name='qemu' type='raw'/>
      <source protocol='iscsi' name='iqn.2013-07.com.
      example:iscsi-nopool/0'>
        <host name='example.com' port='3260'/>
        <initiator>
          <iqn name='iqn.2013-07.com.example:client'/>
        </initiator>
      </source>
      <target dev='sdb' bus='scsi'/>
    </disk>
    <disk type='volume' device='disk'>
      <driver name='qemu' type='raw'/>
      <source pool='iscsi-pool' volume='unit:0:0:1' mode='host'/>
      <target dev='vdb' bus='virtio'/>
    </disk>
    <disk type='volume' device='disk'>
      <driver name='qemu' type='raw'/>
      <source pool='iscsi-pool' volume='unit:0:0:2' mode='direct'/>
      <target dev='vdc' bus='virtio'/>
    </disk>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' queues='4'/>
      <source file='/var/lib/libvirt/images/domain.qcow'/>
      <backingStore type='file'>
        <format type='qcow2'/>
        <source file='/var/lib/libvirt/images/snapshot.qcow'/>
        <backingStore type='block'>
          <format type='raw'/>
          <source dev='/dev/mapper/base'/>
          <backingStore/>
        </backingStore>
      </backingStore>
      <target dev='vdd' bus='virtio'/>
    </disk>
  </devices>

  <!-- Filesystems -->

  <devices>
    <filesystem type='template'>
      <source name='my-vm-template'/>
      <target dir='/'/>
    </filesystem>
    <filesystem type='mount' accessmode='passthrough'>
      <driver type='path' wrpolicy='immediate'/>
      <source dir='/export/to/guest'/>
      <target dir='/import/from/host'/>
      <readonly/>
    </filesystem>
    <filesystem type='file' accessmode='passthrough'>
      <driver name='loop' type='raw'/>
      <driver type='path' wrpolicy='immediate'/>
      <source file='/export/to/guest.img'/>
      <target dir='/import/from/host'/>
      <readonly/>
    </filesystem>
  </devices>

  <!-- Controllers -->

  <devices>
    <controller type='ide' index='0'/>
    <controller type='virtio-serial' index='0' ports='16'
    vectors='4'/>
    <controller type='virtio-serial' index='1'>
```

```
      <address type='pci' domain='0x0000' bus='0x00'
      slot='0x0a'
          function='0x0'/>
    </controller>
    <controller type='scsi' index='0' model='virtio-scsi'>
      <driver iothread='4'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x0b'
          function='0x0'/>
    </controller>
  </devices>
  <devices>
    <controller type='usb' index='0' model='ich9-ehci1'>
      <address type='pci' domain='0' bus='0' slot='4'
      function='7'/>
    </controller>
    <controller type='usb' index='0' model='ich9-uhci1'>
      <master startport='0'/>
      <address type='pci' domain='0' bus='0' slot='4' function='0'
          multifunction='on'/>
    </controller>
  </devices>
  <devices>
    <controller type='pci' index='0' model='pci-root'/>
    <controller type='pci' index='1' model='pci-bridge'>
      <address type='pci' domain='0' bus='0' slot='5' function='0'
          multifunction='off'/>
    </controller>
  </devices>
  <devices>
    <controller type='pci' index='0' model='pcie-root'/>
    <controller type='pci' index='1' model='dmi-to-pci-bridge'>
      <address type='pci' domain='0' bus='0' slot='0xe'
      function='0'/>
    </controller>
    <controller type='pci' index='2' model='pci-bridge'>
      <address type='pci' domain='0' bus='1' slot='1'
      function='0'/>
    </controller>
  </devices>

  <!-- Device leases -->

  <devices>
    ...
    <lease>
      <lockspace>somearea</lockspace>
      <key>somekey</key>
      <target path='/some/lease/path' offset='1024'/>
    </lease>
    ...
  </devices>

  <!-- USB / PCI / SCSI devices -->

  <devices>
    <hostdev mode='subsystem' type='usb'>
      <source startupPolicy='optional'>
        <vendor id='0x1234'/>
        <product id='0xbeef'/>
      </source>
      <boot order='2'/>
    </hostdev>
  </devices>
  <devices>
```

```
    <hostdev mode='subsystem' type='pci' managed='yes'>
      <source>
        <address domain='0x0000' bus='0x06' slot='0x02'
        function='0x0'/>
      </source>
      <boot order='1'/>
      <rom bar='on' file='/etc/fake/boot.bin'/>
    </hostdev>
  </devices>
  <devices>
    <hostdev mode='subsystem' type='scsi' sgio='filtered'
    rawio='yes'>
      <source>
        <adapter name='scsi_host0'/>
        <address bus='0' target='0' unit='0'/>
      </source>
      <readonly/>
      <address type='drive' controller='0' bus='0' target='0'
      unit='0'/>
    </hostdev>
  </devices>
  <devices>
    <hostdev mode='subsystem' type='scsi'>
      <source protocol='iscsi' name='iqn.2014-08.com.
      example:iscsi-nopool/1'>
        <host name='example.com' port='3260'/>
        <auth username='myuser'>
          <secret type='iscsi' usage='libvirtiscsi'/>
        </auth>
      </source>
      <address type='drive' controller='0' bus='0' target='0'
      unit='0'/>
    </hostdev>
  </devices>
  <devices>
    <hostdev mode='subsystem' type='scsi_host'>
      <source protocol='vhost' wwpn='naa.50014057667280d8'/>
    </hostdev>
  </devices>
  <devices>
    <hostdev mode='subsystem' type='mdev' model='vfio-pci'>
    <source>
      <address uuid='c2177883-f1bb-47f0-914d-32a22e3a8804'/>
    </source>
    </hostdev>
    <hostdev mode='subsystem' type='mdev' model='vfio-ccw'>
    <source>
      <address uuid='9063cba3-ecef-47b6-abcf-3fef4fdcad85'/>
    </source>
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0001'/>
    </hostdev>
  </devices>

  <!-- Block / character devices -->

  <hostdev mode='capabilities' type='storage'>
    <source>
      <block>/dev/sdf1</block>
    </source>
  </hostdev>
  <hostdev mode='capabilities' type='misc'>
    <source>
      <char>/dev/input/event3</char>
    </source>
  </hostdev>
```

```
<hostdev mode='capabilities' type='net'>
  <source>
    <interface>eth0</interface>
  </source>
</hostdev>

<!-- Redirected devices -->

<devices>
  <redirdev bus='usb' type='tcp'>
    <source mode='connect' host='localhost' service='4000'/>
    <boot order='1'/>
  </redirdev>
  <redirfilter>
    <usbdev class='0x08' vendor='0x1234' product='0xbeef'
    version='2.56'
        allow='yes'/>
    <usbdev allow='no'/>
  </redirfilter>
</devices>

<!-- Smartcard devices -->

<devices>
  <smartcard mode='host'/>
  <smartcard mode='host-certificates'>
    <certificate>cert1</certificate>
    <certificate>cert2</certificate>
    <certificate>cert3</certificate>
    <database>/etc/pki/nssdb/</database>
  </smartcard>
  <smartcard mode='passthrough' type='tcp'>
    <source mode='bind' host='127.0.0.1' service='2001'/>
    <protocol type='raw'/>
    <address type='ccid' controller='0' slot='0'/>
  </smartcard>
  <smartcard mode='passthrough' type='spicevmc'/>
</devices>
<devices>
  <interface type='direct' trustGuestRxFilters='yes'>
    <source dev='eth0'/>
    <mac address='52:54:00:5d:c7:9e'/>
    <boot order='1'/>
    <rom bar='off'/>
  </interface>
</devices>

<!-- Network interfaces -->

<devices>
  <interface type='network'>
    <source network='default'/>
  </interface>
  ...
  <interface type='network'>
    <source network='default' portgroup='engineering'/>
    <target dev='vnet7'/>
    <mac address="00:11:22:33:44:55"/>
    <virtualport>
      <parameters instanceid='09b11c53-8b5c-4eeb-8f00-
      d84eaa0aaa4f'/>
    </virtualport>
  </interface>
</devices>
```

```
<!-- Bridge to LAN -->

<devices>
  ...
  <interface type='bridge'>
    <source bridge='br0'/>
  </interface>
  <interface type='bridge'>
    <source bridge='br1'/>
    <target dev='vnet7'/>
    <mac address="00:11:22:33:44:55"/>
  </interface>
  <interface type='bridge'>
    <source bridge='ovsbr'/>
    <virtualport type='openvswitch'>
      <parameters profileid='menial'
          interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f'/>
    </virtualport>
  </interface>
  ...
</devices>
<devices>
  ...
  <interface type='bridge'>
    <source bridge='br0'/>
  </interface>
  <interface type='bridge'>
    <source bridge='br1'/>
    <target dev='vnet7'/>
    <mac address="00:11:22:33:44:55"/>
  </interface>
  <interface type='bridge'>
    <source bridge='midonet'/>
    <virtualport type='midonet'>
      <parameters interfaceid='0b2d64da-3d0e-431e-afdd-
      804415d6ebbb'/>
    </virtualport>
  </interface>
  ...
</devices>

<!-- Userspace SLIRP stack -->

<devices>
  <interface type='user'/>
  ...
  <interface type='user'>
    <mac address="00:11:22:33:44:55"/>
    <ip family='ipv4' address='172.17.2.0' prefix='24'/>
    <ip family='ipv6' address='2001:db8:ac10:fd01::'
    prefix='64'/>
  </interface>
</devices>

<!-- Generic ethernet connection -->

<devices>
  <interface type='ethernet'/>
  ...
  <interface type='ethernet'>
    <target dev='vnet7'/>
    <script path='/etc/qemu-ifup-mynet'/>
  </interface>
</devices>
```

```
  <devices>
    ...
    <interface type='direct' trustGuestRxFilters='no'>
      <source dev='eth0' mode='vepa'/>
    </interface>
  </devices>
  <devices>
    ...
    <interface type='direct'>
      <source dev='eth0.2' mode='vepa'/>
      <virtualport type="802.1Qbg">
        <parameters managerid="11" typeid="1193047"
        typeidversion="2"
            instanceid="09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f"/>
      </virtualport>
    </interface>
  </devices>
  <devices>
    ...
    <interface type='direct'>
      <source dev='eth0' mode='private'/>
      <virtualport type='802.1Qbh'>
        <parameters profileid='finance'/>
      </virtualport>
    </interface>
  </devices>
  <devices>
    <interface type='hostdev' managed='yes'>
      <driver name='vfio'/>
      <source>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x07'
            function='0x0'/>
      </source>
      <mac address='52:54:00:6d:90:02'/>
      <virtualport type='802.1Qbh'>
        <parameters profileid='finance'/>
      </virtualport>
    </interface>
  </devices>
  <devices>
    <interface type='mcast'>
      <mac address='52:54:00:6d:90:01'/>
      <source address='230.0.0.1' port='5558'/>
    </interface>
  </devices>
  <devices>
    <interface type='server'>
      <mac address='52:54:00:22:c9:42'/>
      <source address='192.168.0.1' port='5558'/>
    </interface>
    ...
    <interface type='client'>
      <mac address='52:54:00:8b:c9:51'/>
      <source address='192.168.0.1' port='5558'/>
    </interface>
  </devices>
  <devices>
    <interface type='udp'>
      <mac address='52:54:00:22:c9:42'/>
      <source address='127.0.0.1' port='11115'>
        <local address='127.0.0.1' port='11116'/>
      </source>
    </interface>
  </devices>
```

```
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <model type='ne2k_pci'/>
  </interface>
</devices>
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <model type='virtio'/>
    <driver name='vhost' txmode='iothread' ioeventfd='on'
    event_idx='off'
        queues='5' rx_queue_size='256' tx_queue_size='256'>
      <host csum='off' gso='off' tso4='off' tso6='off'
      ecn='off' ufo='off'
          mrg_rxbuf='off'/>
      <guest csum='off' tso4='off' tso6='off' ecn='off'
      ufo='off'/>
    </driver>
    </interface>
</devices>
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <model type='virtio'/>
    <backend tap='/dev/net/tun' vhost='/dev/vhost-net'/>
    <driver name='vhost' txmode='iothread' ioeventfd='on'
    event_idx='off'
        queues='5'/>
    <tune>
      <sndbuf>1600</sndbuf>
    </tune>
  </interface>
</devices>
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
  </interface>
</devices>
<devices>
  <interface type='network'>
    <source network='default'/>
    <guest dev='myeth'/>
  </interface>
</devices>
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <boot order='1'/>
  </interface>
</devices>
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </interface>
</devices>
```

```
  <devices>
    ...
    <interface type='bridge'>
      <source bridge='br0'/>
      <backenddomain name='netvm'/>
    </interface>
    ...
  </devices>
  <devices>
    <interface type='network'>
      <source network='default'/>
      <target dev='vnet0'/>
      <bandwidth>
        <inbound average='1000' peak='5000' floor='200'
        burst='1024'/>
        <outbound average='128' peak='256' burst='256'/>
      </bandwidth>
    </interface>
  </devices>
  <devices>
    <interface type='bridge'>
      <vlan>
        <tag id='42'/>
      </vlan>
      <source bridge='ovsbr0'/>
      <virtualport type='openvswitch'>
        <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-
        d84eaa0aaa4f'/>
      </virtualport>
    </interface>
    <interface type='bridge'>
      <vlan trunk='yes'>
        <tag id='42'/>
        <tag id='123' nativeMode='untagged'/>
      </vlan>
      ...
    </interface>
  </devices>
  <devices>
    <interface type='network'>
      <source network='default'/>
      <target dev='vnet0'/>
      <link state='down'/>
    </interface>
  </devices>
  <devices>
    <interface type='network'>
      <source network='default'/>
      <target dev='vnet0'/>
      <mtu size='1500'/>
    </interface>
  </devices>
  <devices>
    <interface type='network'>
      <source network='default'/>
      <target dev='vnet0'/>
      <coalesce>
        <rx>
          <frames max='7'/>
        </rx>
      </coalesce>
    </interface>
  </devices>
```

```
<!-- IP configuration -->

<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
    <ip address='192.168.122.5' prefix='24'/>
    <ip address='192.168.122.5' prefix='24'
    peer='10.0.0.10'/>
    <route family='ipv4' address='192.168.122.0' prefix='24'
        gateway='192.168.122.1'/>
    <route family='ipv4' address='192.168.122.8'
    gateway='192.168.122.1'/>
  </interface>
  ...
  <hostdev mode='capabilities' type='net'>
    <source>
      <interface>eth0</interface>
    </source>
    <ip address='192.168.122.6' prefix='24'/>
    <route family='ipv4' address='192.168.122.0' prefix='24'
    gateway='192.168.122.1'/>
    <route family='ipv4' address='192.168.122.8'
    gateway='192.168.122.1'/>
  </hostdev>
  ...
</devices>
<devices>
  <interface type='ethernet'>
    <source/>
      <ip address='192.168.123.1' prefix='24'/>
      <ip address='10.0.0.10' prefix='24' peer='192.168.122.5'/>
      <route family='ipv4' address='192.168.42.0' prefix='24'
      gateway='192.168.123.4'/>
    <source/>
    ...
  </interface>
  ...
</devices>

<!-- vhost-user interface -->

<devices>
  <interface type='vhostuser'>
    <mac address='52:54:00:3b:83:1a'/>
    <source type='unix' path='/tmp/vhost1.sock' mode='server'/>
    <model type='virtio'/>
  </interface>
  <interface type='vhostuser'>
    <mac address='52:54:00:3b:83:1b'/>
    <source type='unix' path='/tmp/vhost2.sock'
    mode='client'>
      <reconnect enabled='yes' timeout='10'/>
    </source>
    <model type='virtio'/>
    <driver queues='5'/>
  </interface>
</devices>

<!-- Traffic filtering with NWFilter -->

<devices>
  <interface ...>
    ...
    <filterref filter='clean-traffic'/>
  </interface>
```

```
    <interface ...>
      ...
      <filterref filter='myfilter'>
        <parameter name='IP' value='104.207.129.11'/>
        <parameter name='IP6_ADDR' value='2001:19f0:300:2102::'/>
        <parameter name='IP6_MASK' value='64'/>
        ...
      </filterref>
    </interface>
  </devices>

  <!-- Input devices -->

  <devices>
    <input type='mouse' bus='usb'/>
    <input type='keyboard' bus='usb'/>
    <input type='mouse' bus='virtio'/>
    <input type='keyboard' bus='virtio'/>
    <input type='tablet' bus='virtio'/>
    <input type='passthrough' bus='virtio'>
      <source evdev='/dev/input/event1/>'
    </input>
  </devices>

  <!-- Hub devices -->

  <devices>
    <hub type='usb'/>
  </devices>

  <!-- Graphical framebuffers -->

  <devices>
    <graphics type='sdl' display=':0.0'/>
    <graphics type='vnc' port='5904' sharePolicy='allow-
    exclusive'>
      <listen type='address' address='1.2.3.4'/>
    </graphics>
    <graphics type='rdp' autoport='yes' multiUser='yes' />
    <graphics type='desktop' fullscreen='yes'/>
    <graphics type='spice'>
      <listen type='network' network='rednet'/>
    </graphics>
  </devices>

  <!-- Video devices -->

  <devices>
    <video>
      <model type='vga' vram='16384' heads='1'>
        <acceleration accel3d='yes' accel2d='yes'/>
      </model>
    </video>
  </devices>

  <!-- Consoles, serial, parallel & channel devices -->

  <devices>
    <parallel type='pty'>
      <source path='/dev/pts/2'/>
      <target port='0'/>
    </parallel>
    <serial type='pty'>
      <source path='/dev/pts/3'/>
      <target port='0'/>
    </serial>
```

```
    <serial type='file'>
      <source path='/tmp/file' append='on'>
        <seclabel model='dac' relabel='no'/>
      </source>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <source path='/dev/pts/4'/>
      <target port='0'/>
    </console>
    <channel type='unix'>
      <source mode='bind' path='/tmp/guestfwd'/>
      <target type='guestfwd' address='10.0.2.1' port='4600'/>
    </channel>
  </devices>
  <devices>
    <parallel type='pty'>
      <source path='/dev/pts/2'/>
      <target port='0'/>
    </parallel>
  </devices>
  <devices>
    <!-- Serial port -->
    <serial type='pty'>
      <source path='/dev/pts/3'/>
      <target port='0'/>
    </serial>
  </devices>
  <devices>
    <!-- USB serial port -->
    <serial type='pty'>
      <target type='usb-serial' port='0'>
        <model name='usb-serial'/>
      </target>
      <address type='usb' bus='0' port='1'/>
    </serial>
  </devices>
  <devices>
    <!-- Serial console -->
    <console type='pty'>
      <source path='/dev/pts/2'/>
     <target type='serial' port='0'/>
    </console>
  </devices>
  <devices>
    <!-- KVM virtio console -->
    <console type='pty'>
      <source path='/dev/pts/5'/>
      <target type='virtio' port='0'/>
    </console>
  </devices>
  <devices>
    <console type='pty'>
      <target type='serial'/>
    </console>
    <console type='pty'>
      <target type='virtio'/>
    </console>
  </devices>
  <devices>
    <serial type='pty'/>
  </devices>
```

```
  <devices>
    <console type='pty'/>
  </devices>
  <devices>
    <serial type='pty'/>
    <console type='pty'/>
  </devices>
  <devices>
    <channel type='unix'>
      <source mode='bind' path='/tmp/guestfwd'/>
      <target type='guestfwd' address='10.0.2.1' port='4600'/>
    </channel>
    <!-- KVM virtio channel -->
    <channel type='pty'>
      <target type='virtio' name='arbitrary.virtio.serial.port.
      name'/>
    </channel>
    <channel type='unix'>
      <source mode='bind' path='/var/lib/libvirt/qemu/
      f16x86_64.agent'/>
      <target type='virtio' name='org.qemu.guest_agent.0'
      state='connected'/>
    </channel>
    <channel type='spicevmc'>
      <target type='virtio' name='com.redhat.spice.0'/>
    </channel>
  </devices>
  <devices>
    <console type='stdio'>
      <target port='1'/>
    </console>
  </devices>
```

```
<devices>
  <serial type="file">
    <source path="/var/log/vm/vm-serial.log"/>
    <target port="1"/>
  </serial>
</devices>
<devices>
  <serial type='vc'>
    <target port="1"/>
  </serial>
</devices>
  <devices>
  <serial type='null'>
    <target port="1"/>
  </serial>
</devices>
<devices>
  <serial type="pty">
    <source path="/dev/pts/3"/>
    <target port="1"/>
  </serial>
</devices>
<devices>
  <serial type="dev">
    <source path="/dev/ttyS0"/>
    <target port="1"/>
  </serial>
</devices>
<devices>
  <serial type="pipe">
    <source path="/tmp/mypipe"/>
```

```
      <target port="1"/>
    </serial>
  </devices>
  <devices>
    <serial type="tcp">
      <source mode="connect" host="0.0.0.0" service="2445"/>
      <protocol type="raw"/>
      <target port="1"/>
    </serial>
  </devices>
  <devices>
    <serial type="tcp">
      <source mode="bind" host="127.0.0.1" service="2445"/>
      <protocol type="raw"/>
      <target port="1"/>
    </serial>
  </devices>
  <devices>
    <serial type="tcp">
      <source mode="connect" host="0.0.0.0" service="2445"/>
      <protocol type="telnet"/>
      <target port="1"/>
    </serial>
    ...
    <serial type="tcp">
      <source mode="bind" host="127.0.0.1" service="2445"/>
      <protocol type="telnet"/>
      <target port="1"/>
    </serial>
  </devices>
```

```
<devices>
  <serial type="tcp">
    <source mode='connect' host="127.0.0.1" service="5555"
    tls="yes"/>
    <protocol type="raw"/>
    <target port="0"/>
  </serial>
</devices>
<devices>
  <serial type="udp">
    <source mode="bind" host="0.0.0.0" service="2445"/>
    <source mode="connect" host="0.0.0.0" service="2445"/>
    <target port="1"/>
  </serial>
</devices>
<devices>
  <serial type="unix">
    <source mode="bind" path="/tmp/foo"/>
    <target port="1"/>
  </serial>
</devices>
<devices>
  <serial type="spiceport">
    <source channel="org.qemu.console.serial.0"/>
    <target port="1"/>
  </serial>
</devices>
<devices>
  <serial type="nmdm">
    <source master="/dev/nmdm0A" slave="/dev/nmdm0B"/>
  </serial>
</devices>
```

```
<devices>
  <sound model='es1370'/>
</devices>
<devices>
  <sound model='ich6'>
    <codec type='micro'/>
  </sound>
</devices>
<devices>
  <watchdog model='i6300esb'/>
</devices>
<devices>
  <watchdog model='i6300esb' action='poweroff'/>
</devices>
<devices>
  <memballoon model='virtio'/>
</devices>
<devices>
  <memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02'
        function='0x0'/>
    <stats period='10'/>
    <driver iommu='on' ats='on'/>
  </memballoon>
</devices>
<devices>
  <rng model='virtio'>
    <rate period="2000" bytes="1234"/>
    <backend model='random'>/dev/random</backend>
    <!-- OR -->
    <backend model='egd' type='udp'>
      <source mode='bind' service='1234'/>
      <source mode='connect' host='1.2.3.4' service='1234'/>
    </backend>
  </rng>
</devices>

<!-- TPM device -->

<devices>
  <tpm model='tpm-tis'>
    <backend type='passthrough'>
      <device path='/dev/tpm0'/>
    </backend>
  </tpm>
</devices>
<devices>
  <tpm model='tpm-tis'>
    <backend type='emulator' version='2.0'>
    </backend>
  </tpm>
</devices>

<!-- NVRAM device -->

<devices>
  <nvram>
    <address type='spapr-vio' reg='0x3000'/>
  </nvram>
</devices>
<devices>
  <panic model='hyperv'/>
  <panic model='isa'>
    <address type='isa' iobase='0x505'/>
  </panic>
</devices>
```

```
<!-- Shared memory devic -->

<devices>
  <shmem name='my_shmem0'>
    <model type='ivshmem-plain'/>
    <size unit='M'>4</size>
  </shmem>
  <shmem name='shmem_server'>
    <model type='ivshmem-doorbell'/>
    <size unit='M'>2</size>
    <server path='/tmp/socket-shmem'/>
    <msi vectors='32' ioeventfd='on'/>
  </shmem>
</devices>

<!-- Memory devices -->

<devices>
  <memory model='dimm' access='private' discard='yes'>
    <target>
      <size unit='KiB'>524287</size>
      <node>0</node>
    </target>
  </memory>
  <memory model='dimm'>
    <source>
      <pagesize unit='KiB'>4096</pagesize>
      <nodemask>1-3</nodemask>
    </source>
    <target>
      <size unit='KiB'>524287</size>
      <node>1</node>
    </target>
  </memory>
  <!-- IOMMU devices -->

 <devices>
  <iommu model='intel'>
    <driver intremap='on'/>
  </iommu>
</devices>

  <memory model='nvdimm'>
    <source>
      <path>/tmp/nvdimm</path>
      <alignsize unit='KiB'>2048</alignsize>
    </source>
    <target>
      <size unit='KiB'>524288</size>
      <node>1</node>
      <label>
        <size unit='KiB'>128</size>
      </label>
      <readonly/>
    </target>
  </memory>
  <memory model='nvdimm'>
    <source>
      <path>/dev/dax0.0</path>
      <pmem/>
    </source>
    <target>
      <size unit='KiB'>524288</size>
      <node>1</node>
      <label>
        <size unit='KiB'>128</size>
      </label>
    </target>
  </memory>
</devices>

<!-- IOMMU devices -->

<devices>
  <iommu model='intel'>
    <driver intremap='on'/>
  </iommu>
</devices>

<!-- Vsock -->

<devices>
  <vsock model='virtio'>
    <cid auto='no' address='3'/>
  </vsock>
</devices>

<!-- Security label -->

<seclabel type='dynamic' model='selinux'/>
<seclabel type='dynamic' model='selinux'>
  <baselabel>system_u:system_r:my_svirt_t:s0</baselabel>
</seclabel>
<seclabel type='static' model='selinux' relabel='no'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>
<seclabel type='static' model='selinux' relabel='yes'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>
<seclabel type='none'/>
<!-- Key Wrap -->

<keywrap>
  <cipher name='aes' state='off'/>
</keywrap>

<!-- Launch Security -->

<launchSecurity type='sev'>
  <policy>0x0001</policy>
  <cbitpos>47</cbitpos>
  <reducedPhysBits>1</reducedPhysBits>
  <dhCert>RBBBSDDD=FDDCCCDDDG</dhCert>
  <session>AAACCCDD=FFFCCCDSDS</session>
</launchSecurity>
</domain>
```

The following are some general rules for the schema:

- Some attributes on some elements can be safely omitted.
- Devices not available to the domain can be safely omitted.
- Some devices can be configured in different ways. Use the method that closely matches the device.
- In many cases, libvirt can discover and configure devices on its own. If the discovery and config are correct, there is no need to configure the device via XML.
- It is possible to add attributes and elements later based on device availability.

Example Paravirtualized XEN Schemas

The following are some example XML configurations for Xen guest domains. For full details of the available options, consult the libvirt domain XML documentation.

Paravirtualized Guest Bootloader

Using a bootloader allows a paravirtualized guest to be booted using a kernel stored inside its virtual disk image (Listing A-2).

Listing A-2. Paravirtualized Guest Bootloader

```
<domain type='xen'>
  <name>fc8</name>
  <bootloader>/usr/bin/pygrub</bootloader>
  <os>
    <type>linux</type>
  </os>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <devices>
    <disk type='file'>
      <source file='/var/lib/xen/images/fc4.img'/>
      <target dev='sda1'/>
    </disk>
    <interface type='bridge'>
      <source bridge='xenbr0'/>
      <mac address='aa:00:00:00:00:11'/>
      <script path='/etc/xen/scripts/vif-bridge'/>
    </interface>
    <console tty='/dev/pts/5'/>
  </devices>
</domain>
```

Paravirtualized Guest Direct Kernel Boot

To install paravirtualized guests, it is typical to boot the domain using a kernel and initrd stored in the host OS (Listing A-3).

Listing A-3. Paravirtualized Direct Kernel Boot

```
<domain type='xen'>
  <name>fc8</name>
  <os>
    <type>linux</type>
    <kernel>/var/lib/xen/install/vmlinuz-fedora8-x86_64
    </kernel>
    <initrd>/var/lib/xen/install/initrd-vmlinuz-fedora8-x86_64
    </initrd>
    <cmdline> kickstart=http://example.com/myguest.ks </cmdline>
  </os>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <devices>
    <disk type='file'>
      <source file='/var/lib/xen/images/fc4.img'/>
      <target dev='sda1'/>
    </disk>
    <interface type='bridge'>
      <source bridge='xenbr0'/>
      <mac address='aa:00:00:00:00:11'/>
      <script path='/etc/xen/scripts/vif-bridge'/>
    </interface>
    <graphics type='vnc' port='-1'/>
    <console tty='/dev/pts/5'/>
  </devices>
</domain>
```

Fully Virtualized Guest BIOS Boot

Fully virtualized guests use the emulated BIOS to boot off the primary hard disk, CD-ROM, or network PXE ROM (Listing A-4) .

Listing A-4. Fully Virtualized Guest BIOS Boot

```
<domain type='xen' id='3'>
  <name>fv0</name>
  <uuid>4dea22b31d52d8f32516782e98ab3fa0</uuid>
  <os>
    <type>hvm</type>
    <loader>/usr/lib/xen/boot/hvmloader</loader>
    <boot dev='hd'/>
  </os>
  <memory>524288</memory>
  <vcpu>1</vcpu>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <features>
    <pae/>
    <acpi/>
    <apic/>
  </features>
  <clock sync="localtime"/>
  <devices>
    <emulator>/usr/lib/xen/bin/qemu-dm</emulator>
    <interface type='bridge'>
      <source bridge='xenbr0'/>
      <mac address='00:16:3e:5d:c7:9e'/>
      <script path='vif-bridge'/>
    </interface>
    <disk type='file'>
      <source file='/var/lib/xen/images/fv0'/>
      <target dev='hda'/>
    </disk>
    <disk type='file' device='cdrom'>
      <source file='/var/lib/xen/images/fc5-x86_64-boot.iso'/>
      <target dev='hdc'/>
      <readonly/>
    </disk>
    <disk type='file' device='floppy'>
      <source file='/root/fd.img'/>
      <target dev='fda'/>
    </disk>
    <graphics type='vnc' port='5904'/>
  </devices>
</domain>
```

Fully Virtualized Guest Direct Kernel Boot

With Xen 3.2.0 or newer, it is possible to bypass the BIOS and directly boot a Linux kernel and initrd as a fully virtualized domain (Listing A-5). This allows for complete automation of the OS installation, for example, using the Anaconda kickstart support to install the domain.

Listing A-5. Fully Virtualized Guest Direct Kernel Boot

```
<domain type='xen' id='3'>
  <name>fv0</name>
  <uuid>4dea22b31d52d8f32516782e98ab3fa0</uuid>
  <os>
    <type>hvm</type>
    <loader>/usr/lib/xen/boot/hvmloader</loader>
    <kernel>/var/lib/xen/install/vmlinuz-fedora8-x86_64</kernel>
    <initrd>/var/lib/xen/install/initrd-vmlinuz-
    fedora8-x86_64</initrd>
    <cmdline> kickstart=http://example.com/myguest.ks </cmdline>
  </os>
  <memory>524288</memory>
  <vcpu>1</vcpu>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <features>
    <pae/>
    <acpi/>
    <apic/>
  </features>
  <clock sync="localtime"/>
  <devices>
    <emulator>/usr/lib/xen/bin/qemu-dm</emulator>
    <interface type='bridge'>
      <source bridge='xenbr0'/>
      <mac address='00:16:3e:5d:c7:9e'/>
      <script path='vif-bridge'/>
    </interface>
    <disk type='file'>
      <source file='/var/lib/xen/images/fv0'/>
      <target dev='hda'/>
    </disk>
    <disk type='file' device='cdrom'>
      <source file='/var/lib/xen/images/fc5-x86_64-boot.iso'/>
      <target dev='hdc'/>
      <readonly/>
    </disk>
    <disk type='file' device='floppy'>
      <source file='/root/fd.img'/>
      <target dev='fda'/>
    </disk>
    <graphics type='vnc' port='5904'/>
  </devices>
</domain>
```

Example Paravirtualized KVM and Qemu Schemas

Paravirtualized schemas are also supported by specifying the emulation to be used.

QEMU Emulated Guest on x86_64

The XML in Listing A-6 specifies that the QEMU emulator is to be used to support the domain.

Listing A-6. QEMU Emulated Guest on x86_64

```
<domain type='qemu'>
  <name>QEMU-fedora-i686</name>
  <uuid>c7a5fdbd-cdaf-9455-926a-d65c16db1809</uuid>
  <memory>219200</memory>
  <currentMemory>219200</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='cdrom'/>
  </os>
  <devices>
```

```
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='cdrom'>
      <source file='/home/user/boot.iso'/>
      <target dev='hdc'/>
      <readonly/>
    </disk>
    <disk type='file' device='disk'>
      <source file='/home/user/fedora.img'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
    </interface>
    <graphics type='vnc' port='-1'/>
  </devices>
</domain>
```

KVM Hardware Accelerated Guest on i686

Listing A-7 specifies KVM as the emulator for the guest domain.

Listing A-7. KVM Hardware-Accelerated Guest on i686

```
<domain type='kvm'>
  <name>demo2</name>
  <uuid>4dea24b3-1d52-d8f3-2516-782e98a23fa0</uuid>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch="i686">hvm</type>
  </os>
  <clock sync="localtime"/>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/demo2.img'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
      <mac address='24:42:53:21:52:45'/>
    </interface>
    <graphics type='vnc' port='-1' keymap='de'/>
  </devices>
</domain>
```

The Networks Schema

The root element required for all virtual networks is named network and has no configurable attributes. (However, since 0.10.0, there is one optional read-only attribute; when examining the live configuration of a network, the attribute connections, if present, specify the number of guest interfaces currently connected via this network.) The network XML format has been available since 0.3.0.

The Network XML schema consists of a number of XML elements. The documentation for this schema is extensive but in most cases the uses of the elements in the schema are obvious. If you need documentation, refer to https://libvirt.org/formatnetwork.html for more specific information on each element and attribute.

The Network Schema Details

Listing A-8 shows the first elements that provide basic metadata about the virtual network.

Listing A-8. General Metadata

```
<network ipv6='yes' trustGuestRxFilters='no'>
  <name>default</name>
  <uuid>3e3fce45-4f53-4fa7-bb32-11f34168b82b</uuid>
  <metadata>
    <app1:foo xmlns:app1="http://app1.org/app1/">..</app1:foo>
    <app2:bar xmlns:app2="http://app1.org/app2/">..</app2:bar>
  </metadata>
```

The Connectivity Schema

The next set of elements control how a virtual network is provided with connectivity to the physical LAN (if at all). See Listing A-9.

Listing A-9. Connectivity

```
  <bridge name="virbr0" stp="on" delay="5"
  macTableManager="libvirt"/>
<mtu size="9000"/>
<domain name="example.com" localOnly="no"/>
<forward mode="nat" dev="eth0"/>
```

Listing A-10 shows how to specify network forwarding.

Listing A-10. Forwarding

```
  <forward mode='passthrough'>
    <interface dev='eth10'/>
    <interface dev='eth11'/>
    <interface dev='eth12'/>
    <interface dev='eth13'/>
    <interface dev='eth14'/>
  </forward>
```

Listing A-11 shows how to specify network passthrough.

Listing A-11. Passthrough with a Specific Device

```
<forward mode='passthrough'>
  <pf dev='eth0'/>
</forward>
```

Listing A-12 shows how to specify forwarding for a specific driver.

Listing A-12. Forwarding Using a Specific Driver

```
<forward mode='hostdev' managed='yes'>
  <driver name='vfio'/>
  <address type='pci' domain='0' bus='4' slot='0' function='1'/>
  <address type='pci' domain='0' bus='4' slot='0' function='2'/>
  <address type='pci' domain='0' bus='4' slot='0' function='3'/>
</forward>
```

Listing A-13 shows how to specify forwarding for a managed device.

Listing A-13. Forwarding with a Managed Device

```
<forward mode='hostdev' managed='yes'>
  <pf dev='eth0'/>
</forward>
```

Listing A-14 shows how to specify forwarding with a specific device with limits.

Listing A-14. Forwarding a Specific Device with Limits

```
  <forward mode='nat' dev='eth0'/>
  <bandwidth>
    <inbound average='1000' peak='5000' burst='5120'/>
    <outbound average='128' peak='256' burst='256'/>
  </bandwidth>
```

Listing A-15 shows how to forward a specific device with limits.

Listing A-15. Forwarding a Specific Device with Limits

```
  <forward mode='nat' dev='eth0'/>
  <bandwidth>
    <inbound average='1000' peak='5000' burst='5120'/>
    <outbound average='128' peak='256' burst='256'/>
  </bandwidth>
```

Setting a VLAN Tag (on Supported Network Types Only)

Listing A-16 shows how to set the VLAN tag on networks. The VLAN tag specifies how the virtual network interacts with the main host's real network connection.

Listing A-16. Setting a VLAN Tag

```
<network>
  <name>ovs-net</name>
  <forward mode='bridge'/>
  <bridge name='ovsbr0'/>
  <virtualport type='openvswitch'>
```

```
    <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-
    d84eaa0aaa4f'/>
  </virtualport>
  <vlan trunk='yes'>
    <tag id='42' nativeMode='untagged'/>
    <tag id='47'/>
  </vlan>
  <portgroup name='dontpanic'>
    <vlan>
      <tag id='42'/>
    </vlan>
  </portgroup>
</network>
```

Portgroups

Listing A-17 shows the port interaction with the host's ports.

Listing A-17. Portgroups

```
<network>
  <name>ovs-net</name>
  <forward mode='bridge'/>
  <bridge name='ovsbr0'/>
  <virtualport type='openvswitch'>
    <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-
    d84eaa0aaa4f'/>
  </virtualport>
  <vlan trunk='yes'>
    <tag id='42' nativeMode='untagged'/>
    <tag id='47'/>
  </vlan>
  <portgroup name='dontpanic'>
    <vlan>
      <tag id='42'/>
    </vlan>
  </portgroup>
</network>
```

Static Routes

Listing A-18 shows how static routes can interface with the host's real network.

Listing A-18. Static Routes

```
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.128" end="192.168.122.254"/>
    </dhcp>
  </ip>
  <route address="192.168.222.0" prefix="24"
  gateway="192.168.122.2"/>
  <ip family="ipv6" address="2001:db8:ca2:2::1" prefix="64"/>
  <route family="ipv6" address="2001:db8:ca2:3::" prefix="64"
  gateway="2001:db8:ca2:2::2"/>
  <route family="ipv6" address="2001:db9:4:1::" prefix="64"
  gateway="2001:db8:ca2:2::3" metric='2'/>
```

Addressing

Listing A-19 shows how to set up static routes to the real network.

Listing A-19. Static Routes

```
<mac address='00:16:3E:5D:C7:9E'/>
<domain name="example.com"/>
<dns>
  <txt name="example" value="example value"/>
  <forwarder addr="8.8.8.8"/>
  <forwarder domain='example.com' addr="8.8.4.4"/>
  <forwarder domain='www.example.com'/>
  <srv service='name' protocol='tcp' domain='test-domain-
  name' target='.'
    port='1024' priority='10' weight='10'/>
  <host ip='192.168.122.2'>
    <hostname>myhost</hostname>
    <hostname>myhostalias</hostname>
  </host>
</dns>
<ip address="192.168.122.1" netmask="255.255.255.0"
localPtr="yes">
  <dhcp>
    <range start="192.168.122.100" end="192.168.122.254"/>
    <host mac="00:16:3e:77:e2:ed" name="foo.example.com"
    ip="192.168.122.10"/>
    <host mac="00:16:3e:3e:a9:1a" name="bar.example.com"
    ip="192.168.122.11"/>
  </dhcp>
</ip>
<ip family="ipv6" address="2001:db8:ca2:2::1" prefix="64"
localPtr="yes"/>
<route family="ipv6" address="2001:db9:ca1:1::" prefix="64"
 gateway="2001:db8:ca2:2::2"/>
```

Example Configurations

The following sections review some of the more frequently used network configurations.

NAT-Based Network

This example is the so-called “default” virtual network. It is provided and enabled out of the box for all libvirt installations. This is a configuration that allows the guest OS to get outbound connectivity regardless of whether the host uses Ethernet, wireless, dial-up, or VPN networking without requiring any specific admin configuration. In the absence of host networking, it at least allows guests to talk directly to each other. See Listing A-20.

Listing A-20. The Default Virtual Network Configuration

```
<network>
  <name>default</name>
  <bridge name="virbr0"/>
  <forward mode="nat"/>
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.2" end="192.168.122.254"/>
    </dhcp>
  </ip>
  <ip family="ipv6" address="2001:db8:ca2:2::1" prefix="64"/>
</network
```

Listing A-21 is a variation of the previous example that adds an IPv6 DHCP range definition.

Listing A-21. A Variation of the Default Network Configuration

```
<network>
  <name>default</name>
  <bridge name="virbr0"/>
  <forward mode="nat"/>
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.2" end="192.168.122.254"/>
    </dhcp>
  </ip>
  <ip family="ipv6" address="2001:db8:ca2:2::1" prefix="64"/>
</network
```

Routed Network Config

This is a variant on the default network that routes traffic from the virtual network to the LAN without applying any NAT. It requires that the IP address range be preconfigured in the routing tables of the router on the host network. Listing A-22 further specifies that guest traffic may go out only via the `eth1` host network device.

Listing A-22. A Routed Default Network Configuration

```
<network>
  <name>local</name>
  <bridge name="virbr1"/>
  <forward mode="route" dev="eth1"/>
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.2" end="192.168.122.254"/>
    </dhcp>
  </ip>
  <ip family="ipv6" address="2001:db8:ca2:2::1" prefix="64"/>
</network>
```

Listing A-23 is another IPv6 variation. Instead of a DHCP range being specified, this example has a couple of IPv6 host definitions. Note that most of the DHCP host definitions use an id (client ID or DUID) since this has proven to be a more reliable way of specifying the interface and its association with an IPv6 address. The first is a DUID-LLT, the second a DUID-LL, and the third a DUID-UUID. Since 1.0.3.

Listing A-23. A Routed IPv6 Network Configuration

```
<network>
  <name>local6</name>
  <bridge name="virbr1"/>
  <forward mode="route" dev="eth1"/>
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.2" end="192.168.122.254"/>
    </dhcp>
  </ip>
  <ip family="ipv6" address="2001:db8:ca2:2::1" prefix="64">
    <dhcp>
      <host name="paul" ip="2001:db8:ca2:2:3::1"/>
      <host id="0:1:0:1:18:aa:62:fe:0:16:3e:44:55:66"
      ip="2001:db8:ca2:2:3::2"/>
      <host id="0:3:0:1:0:16:3e:11:22:33" name="ralph"
      ip="2001:db8:ca2:2:3::3"/>
      <host id="0:4:7e:7d:f0:7d:a8:bc:c5:d2:13:32:11:ed:16:
      ea:84:63"
        name="badbob" ip="2001:db8:ca2:2:3::4"/>
    </dhcp>
  </ip>
</network>
```

Listing A-24 is yet another IPv6 variation. This variation has only IPv6 defined with DHCPv6 on the primary IPv6 network. A static link is defined for a second IPv6 network, which will not be directly visible on the bridge interface, but there will be a static route defined for this network via the specified gateway. Note that the gateway address must be directly reachable via (on the same subnet as) one of the `<ip>` addresses defined for this `<network>`. Since 1.0.6.

Listing A-24. Another Routed IPv6 Network Configuration

```
<network>
  <name>net7</name>
  <bridge name="virbr7"/>
  <forward mode="route"/>
  <ip family="ipv6" address="2001:db8:ca2:7::1" prefix="64">
    <dhcp>
      <range start="2001:db8:ca2:7::100" end="2001:db8:ca2::1ff"/>
      <host id="0:4:7e:7d:f0:7d:a8:bc:c5:d2:13:32:11:ed:16:
      ea:84:63"
        name="lucas" ip="2001:db8:ca2:2:3::4"/>
    </dhcp>
  </ip>
  <route family="ipv6" address="2001:db8:ca2:8::" prefix="64"
  gateway="2001:db8:ca2:7::4"/>
</network>
```

Isolated Network Config

This variant provides a completely isolated private network for guests. The guests can talk to each other and the host OS but cannot reach any other machines on the LAN because of the omission of the forward element in the XML description. Listing A-25 shows an entire class C network being specified, but it is possible to start with a smaller range.

Listing A-25. Isolated Network

```
<network>
  <name>private</name>
  <bridge name="virbr2"/>
  <ip address="192.168.152.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.152.2" end="192.168.152.254"/>
    </dhcp>
  </ip>
  <ip family="ipv6" address="2001:db8:ca2:3::1" prefix="64"/>
</network>
```

Isolated IPv6 Network Config

This variation of an isolated network defines only IPv6. Note that most of the DHCP host definitions use an ID (a client ID or DUID) since this has proven to be a more reliable way of specifying the interface and its association with an IPv6 address (Listing A-26). The first is a DUID-LLT, the second a DUID-LL, and the third a DUID-UUID. Since version 1.0.3.

Listing A-26. Isolated IPv6 Network

```
<network>
  <name>sixnet</name>
  <bridge name="virbr6"/>
  <ip family="ipv6" address="2001:db8:ca2:6::1" prefix="64">
    <dhcp>
      <host name="peter" ip="2001:db8:ca2:6:6::1"/>
      <host id="0:1:0:1:18:aa:62:fe:0:16:3e:44:55:66"
      ip="2001:db8:ca2:6:6::2"/>
      <host id="0:3:0:1:0:16:3e:11:22:33" name="dariusz"
      ip="2001:db8:ca2:6:6::3"/>
      <host id="0:4:7e:7d:f0:7d:a8:bc:c5:d2:13:32:11:ed:16:
      ea:84:63"
        name="anita" ip="2001:db8:ca2:6:6::4"/>
    </dhcp>
  </ip>
</network>
```

Using an Existing Host Bridge

Since 0.9.4.

Listing A-27 shows how to use a preexisting host bridge, called `br0`. The guests will effectively be directly connected to the physical network. (In other words, their IP addresses will all be on the subnet of the physical network, and there will be no restrictions on inbound or outbound connections.)

Listing A-27. Using an Existing Host Bridge

```
<network>
  <name>host-bridge</name>
  <forward mode="bridge"/>
  <bridge name="br0"/>
</network>
```

Using a macvtap "Direct" Connection

Since version 0.9.4, QEMU and KVM requires Linux kernel 2.6.34 or newer. Listing A-28 shows how to use macvtap to connect to the physical network directly through one of a group of physical devices (without using a host bridge device). As with the host bridge network, the guests will effectively be directly connected to the physical network so their IP addresses will all be on the subnet of the physical network, and there will be no restrictions on inbound or outbound connections. Note that because of a limitation in the implementation of macvtap, these connections do not allow communication directly between the host and the guests. If you require this, you will either need the attached physical switch to be operating in a mirroring mode (so that all traffic coming to the switch is reflected back to the host's interface) or provide alternate means for this communication (e.g., a second interface on each guest that is connected to an isolated network). The other forward modes that use macvtap (private, vepa, and passthrough) would be used in a similar fashion.

Listing A-28. Using a macvtap "Direct" Connection

```
<network>
  <name>direct-macvtap</name>
  <forward mode="bridge">
    <interface dev="eth20"/>
    <interface dev="eth21"/>
    <interface dev="eth22"/>
    <interface dev="eth23"/>
    <interface dev="eth24"/>
  </forward>
</network>
```

Network Config with No Gateway Address

A valid network definition can contain no IPv4 or IPv6 addresses. Such a definition can be used for a "very private" or "very isolated" network since it will not be possible to communicate with the virtualization host via this network. However, this virtual network interface can be used for communication between virtual guest systems. This works for IPv4 and (since 1.0.1) IPv6. However, `ipv6='yes'` must be added for guest-to-guest IPv6 communication (Listing A-29).

Listing A-29. Network Config with No Gateway Address

```
<network ipv6='yes'>
  <name>nogw</name>
  <uuid>7a3b7497-1ec7-8aef-6d5c-38dff9109e93</uuid>
  <bridge name="virbr2" stp="on" delay="0"/>
  <mac address='00:16:3E:5D:C7:9E'/>
</network>
```

Storage Pool XML

Although all storage pool back ends share the same public APIs and XML format, they have varying levels of capabilities. Some may allow the creation of volumes; others may allow only the use of preexisting volumes. Some may have constraints on volume size or placement.

The top-level tag for a storage pool document is `pool`. It has a single attribute type, which is one of `dir`, `fs`, `netfs`, `disk`, `iscsi`, `logical`, `scsi` (all since 0.4.1), `mpath` (since 0.7.1), `rbd` (since 0.9.13), `sheepdog` (since 0.10.0), `gluster` (since 1.2.0), `zfs` (since 1.2.8), or `vstorage` (since 3.1.0). This corresponds to the storage back-end drivers listed later in this appendix.

General Metadata

Listing A-30 shows how to specify the metadata for a storage pool.

Listing A-30. Storage Pool Metadata

```
<pool type="iscsi">
  <name>virtimages</name>
  <uuid>3e3fce45-4f53-4fa7-bb32-11f34168b82b</uuid>
  <allocation>10000000</allocation>
  <capacity>50000000</capacity>
  <available>40000000</available>
```

Source Elements

The following examples show how to specify the source elements for a storage pool.

Specifically, Listing A-31 shows how to specify an iSCSI device for the storage pool.

Listing A-31. Using an iSCSI Device for the Storage Pool

```
<source>
  <host name="iscsi.example.com"/>
  <device path="iqn.2013-06.com.example:iscsi-pool"/>
  <auth type='chap' username='myname'>
    <secret usage='mycluster_myname'/>
  </auth>
  <vendor name="Acme"/>
  <product name="model"/>
</source>
```

Listing A-32 shows how to specify a file path for the storage pool.

Listing A-32. Using a File Path for the Storage Pool

```
<source>
  <device path='/dev/mapper/mpatha' part_separator='no'/>
  <format type='gpt'/>
</source>
```

Listing A-33 shows how to use a SCSI device for the storage pool.

Listing A-33. Using a SCSI Device for the Storage Pool

```
<source>
  <adapter type='scsi_host' name='scsi_host1'/>
</source>
```

Listing A-34 shows how to use a SCSI LUN for the storage pool.

Listing A-34. Using a SCSI LUN for the Storage Pool

```
<source>
  <adapter type='scsi_host'>
    <parentaddr unique_id='1'>
      <address domain='0x0000' bus='0x00' slot='0x1f' addr='0x2'/>
    </parentaddr>
  </adapter>
</source>
```

Listing A-35 shows how to use a SCSI host for the storage pool.

Listing A-35. Using a SCSI Host for the Storage Pool

```
<source>
  <adapter type='fc_host' parent='scsi_host5'
  wwnn='20000000c9831b4b'        wwpn='10000000c9831b4b'/>
</source>
```

Target Elements

A single target element is contained within the top-level pool element for some types of pools (pool types `dir`, `fs`, `netfs`, `logical`, `disk`, `iscsi`, `scsi`, `mpath`, `zfs`). This tag is used to describe the mapping of the storage pool into the host filesystem. It can contain the child elements shown in Listing A-36.

Listing A-36. Specifying a Single Target in the Top-Level Element

```
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <owner>107</owner>
      <group>107</group>
      <mode>0744</mode>
      <label>virt_image_t</label>
    </permissions>
  </target>
</pool>
```

Storage Volume XML

A storage volume will generally be either a file or a device node; since 1.2.0, an optional output-only attribute type lists the actual type (`file`, `block`, `dir`, `network`, `netdir`, or `ploop`), which is also available from `virStorageVolGetInfo()`. The storage volume XML format has been available since 0.4.1.

Storage Volume XML

A storage volume will generally be either a file or a device node; since 1.2.0, an optional output-only attribute type lists the actual type (`file`, `block`, `dir`, `network`, `netdir`, or `ploop`), which is also available from `virStorageVolGetInfo()`. The storage volume XML format has been available since 0.4.1 (Listing A-37).

Listing A-37. How to Specify a Storage Volume

```
  <volume type='file'>
  <name>sparse.img</name>
  <key>/var/lib/xen/images/sparse.img</key>
  <allocation>0</allocation>
  <capacity unit="T">1</capacity>
```

Target Elements

A single target element is contained within the top-level volume element. This tag is used to describe the mapping of the storage volume into the host file system. It can contain the child elements in Listing A-38.

Listing A-38. Specifying a Single Target Element for the Storage Volume

```
<target>
  <path>/var/lib/virt/images/sparse.img</path>
  <format type='qcow2'/>
  <permissions>
    <owner>107</owner>
    <group>107</group>
    <mode>0744</mode>
    <label>virt_image_t</label>
  </permissions>
  <timestamps>
    <atime>1341933637.273190990</atime>
    <mtime>1341930622.047245868</mtime>
    <ctime>1341930622.047245868</ctime>
  </timestamps>
  <encryption type='...'>
    ...
  </encryption>
  <compat>1.1</compat>
  <nocow/>
  <features>
    <lazy_refcounts/>
  </features>
</target>
```

Backing Store Element

A single backingStore element is contained within the top-level volume element. This tag is used to describe the optional copy-on-write, backing store for the storage volume. It can contain the child elements in Listing A-39.

Listing A-39. Specifying a Single Backing Store for the Storage Volume

```
  <backingStore>
    <path>/var/lib/virt/images/master.img</path>
    <format type='raw'/>
    <permissions>
      <owner>107</owner>
      <group>107</group>
      <mode>0744</mode>
      <label>virt_image_t</label>
    </permissions>
  </backingStore>
</volume>
```

Volume Examples

The following examples show some complete examples for specifying a storage volume.

Listing A-40 specifies a file path for the storage volume.

Listing A-40. Specifying a File Path for the Storage Volume

```
<pool type="dir">
  <name>virtimages</name>
  <target>
    <path>/var/lib/virt/images</path>
  </target>
</pool>
```

Listing A-41 specifies an iSCSI device for the storage volume.

Listing A-41. Specifying an iSCSI Device for the Storage Volume

```
<pool type="iscsi">
  <name>virtimages</name>
  <source>
    <host name="iscsi.example.com"/>
    <device path="iqn.2013-06.com.example:iscsi-pool"/>
    <auth type='chap' username='myuser'>
      <secret usage='libvirtiscsi'/>
    </auth>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

Listing A-42 specifies an ummounted file image for the storage volume.

Listing A-42. Specifying a File Image for the Storage Volume

```
<volume>
  <name>sparse.img</name>
  <allocation>0</allocation>
  <capacity unit="T">1</capacity>
  <target>
    <path>/var/lib/virt/images/sparse.img</path>
    <permissions>
      <owner>107</owner>
      <group>107</group>
      <mode>0744</mode>
      <label>virt_image_t</label>
    </permissions>
  </target>
</volume>
```

Listing A-43 specifies an encrypted storage file for the storage volume.

Listing A-43. Specifying an Encrypted File Image for the Storage Volume

```
<volume>
  <name>MyLuks.img</name>
  <capacity unit="G">5</capacity>
  <target>
    <path>/var/lib/virt/images/MyLuks.img</path>
    <format type='raw'/>
    <encryption format='luks'>
      <secret type='passphrase' uuid='f52a81b2-424e-490c-823d-
      6bd4235bc572'/>
    </encryption>
  </target>
</volume>
```

Element and Attribute Overview

As new virtualization engine support gets added to libvirt and to handle cases like QEMU supporting a variety of emulations, a query interface has been added in 0.2.1. This query interface allows you to list the set of supported virtualization capabilities on the host.

```
connect.GetCapabilities()
```

It should be noted at this point that the following XML is returned from libvirt APIs and not passed to it. However, that does not prevent you from modifying this XML and then passing it back to libvirt to change the existing configuration.

Host Capabilities

Listing A-44 provides an example of all the possible elements of `<host/>`.

Listing A-44. Specifying the Capabilities of the Main Host System

```
<capabilities>
  <host>
    <cpu>
      <arch>x86_64</arch>
      <features>
        <vmx/>
      </features>
      <model>core2duo</model>
      <vendor>Intel</vendor>
      <topology sockets="1" cores="2" threads="1"/>
      <feature name="lahf_lm"/>
      <feature name='xtpr'/>
      ...
    </cpu>
    <power_management>
      <suspend_mem/>
      <suspend_disk/>
      <suspend_hybrid/>
    </power_management>
  </host>
</capabilities>
```

Guest Capabilities

Listing A-45 provides an example of all the possible elements of `<guest/>`.

Listing A-45. Specifying a Guest Domain's Capabilities

```
<capabilities>
  <guest>
    <os_type>hvm</os_type>
    <arch name="i686">
      <wordsize>32</wordsize>
      <domain type="xen"></domain>
      <emulator>/usr/lib/xen/bin/qemu-dm</emulator>
      <machine>pc</machine>
      <machine>isapc</machine>
      <loader>/usr/lib/xen/boot/hvmloader</loader>
    </arch>
    <features>
      <cpuselection/>
      <deviceboot/>
    </features>
  </guest>
</capabilities>
```

Node Device XML Format

There are several libvirt functions, all with the prefix `virNodeDevice`, that deal with the management of host devices that can be handed to guests via passthrough as `<hostdev>` elements in the domain XML. These devices are represented as a hierarchy, where a device on a bus has a parent of the bus controller device; the root of the hierarchy is the node named `computer`.

When represented in XML, a node device uses the top-level device element, with the elements in Listing A-46 present according to the type of device.

Listing A-46. Specifying the Node Device Capabilities

```
<device>
  <name>computer</name>
  <capability type='system'>
    <product>2241B36</product>
    <hardware>
      <vendor>LENOVO</vendor>
      <version>ThinkPad T500</version>
      <serial>R89055N</serial>
      <uuid>c9488981-5049-11cb-9c1c-993d0230b4cd</uuid>
    </hardware>
    <firmware>
      <vendor>LENOVO</vendor>
      <version>6FET82WW (3.12 )</version>
      <release_date>11/26/2009</release_date>
    </firmware>
  </capability>
</device>

<device>
  <name>net_eth1_00_27_13_6a_fe_00</name>
  <parent>pci_0000_00_19_0</parent>
  <capability type='net'>
    <interface>eth1</interface>
    <address>00:27:13:6a:fe:00</address>
    <capability type='80203'/>
  </capability>
</device>
```

```
<device>
  <name>pci_0000_02_00_0</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:02:00.0</path>
  <parent>pci_0000_00_04_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>2</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection
    </product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='virt_functions'>
      <address domain='0x0000' bus='0x02' slot='0x10'
      function='0x0'/>
      <address domain='0x0000' bus='0x02' slot='0x10'
      function='0x2'/>
      <address domain='0x0000' bus='0x02' slot='0x10'
      function='0x4'/>
      <address domain='0x0000' bus='0x02' slot='0x10'
      function='0x6'/>
      <address domain='0x0000' bus='0x02' slot='0x11'
      function='0x0'/>
      <address domain='0x0000' bus='0x02' slot='0x11'
      function='0x2'/>
      <address domain='0x0000' bus='0x02' slot='0x11'
      function='0x4'/>
    </capability>
    <iommuGroup number='12'>
      <address domain='0x0000' bus='0x02' slot='0x00'
      function='0x0'/>
      <address domain='0x0000' bus='0x02' slot='0x00'
      function='0x1'/>
    </iommuGroup>
    <pci-express>
      <link validity='cap' port='1' speed='2.5' width='1'/>
      <link validity='sta' speed='2.5' width='1'/>
    </pci-express>
  </capability>
</device>
```

Snapshot XML Format

There are several types of snapshots.

- **Disk snapshot**: The contents of disks (whether a subset or all disks associated with the domain) are saved at a given point of time and can be restored to that state. On a running guest, a disk snapshot is likely to be only crash-consistent rather than clean. (That is, it represents the state of the disk on a sudden power outage and may need fsck or journal replays to be made consistent.) On an inactive guest, a disk snapshot is clean if the disks were clean when the guest was last shut down. Disk snapshots exist in two forms: internal (file formats such as qcow2 track both the snapshot and changes since the snapshot in a single file) and external (the snapshot is one file, and the changes since the snapshot are in another file).

- **Memory state (or VM state)**: This tracks only the state of RAM and all other resources in use by the VM. If the disks are unmodified between the time a VM state snapshot is taken and restored, then the guest will resume in a consistent state; but if the disks are modified externally in the meantime, this is likely to lead to data corruption.
- **System checkpoint**: This is a combination of disk snapshots for all disks as well as VM memory state, which can be used to resume the guest from where it left off with symptoms similar to hibernation (that is, TCP connections in the guest may have timed out, but no files or processes are lost).

libvirt can manage all three types of snapshots. For now, VM state (memory) snapshots are created only by the `save()`, `saveFlags`, and `managedSave()` methods, and they are restored via the `restore()`, `create()`, and `createWithFlags()` methods (as well as via domain autostart). With managed snapshots, libvirt tracks all information internally; with save images, the user tracks the snapshot file, but libvirt provides functions such as `saveImageGetXMLDesc()` to work with those files.

System checkpoints are created by `snapshotCreateXML()` with no flags, and disk snapshots are created by the same function with the `VIR_DOMAIN_SNAPSHOT_CREATE_DISK_ONLY` flag; in both cases, they are restored by the `revertToSnapshot()` function. For these types of snapshots, libvirt tracks each snapshot as a separate `snapshotPtr` object and maintains a tree relationship of which snapshots descended from an earlier point in time.

Attributes of libvirt snapshots are stored as child elements of the `domainsnapshot` element. At snapshot creation time, normally only the name, description, and disks elements are settable; the rest of the fields are ignored on creation and will be filled in by libvirt for informational purposes by snapshotGetXMLDesc(). However, when redefining a snapshot (since 0.9.5), with the VIR_DOMAIN_SNAPSHOT_CREATE_REDEFINE flag of snapshotCreateXML(), all of the XML described here is relevant.

Snapshots are maintained in a hierarchy. A domain can have a current snapshot, which is the most recent snapshot compared to the current state of the domain (although a domain might have snapshots without a current snapshot, if snapshots have been deleted in the meantime). Creating or reverting to a snapshot sets that snapshot as current, and the prior current snapshot is the parent of the new snapshot. Branches in the hierarchy can be formed by reverting to a snapshot with a child and then creating another snapshot.

Snapshot XML Samples

Listing A-47 uses XML to create a disk snapshot of just vda on a qemu domain with two disks.

Listing A-47. An Example of a Disk Snapshot

```
<domainsnapshot>
  <description>Snapshot of OS install and updates</description>
  <disks>
    <disk name='/path/to/old'>
      <source file='/path/to/new'/>
    </disk>
    <disk name='vdb' snapshot='no'/>
  </disks>
</domainsnapshot>
```

Listing A-48 shows the result in XML similar to this from snapshotGetXMLDesc().

Listing A-48. Sample Guest Domain Snapshot

```
<domainsnapshot>
  <name>1270477159</name>
  <description>Snapshot of OS install and updates</description>
  <state>running</state>
  <creationTime>1270477159</creationTime>
  <parent>
    <name>bare-os-install</name>
  </parent>
  <memory snapshot='no'/>
  <disks>
    <disk name='vda' snapshot='external'>
      <driver type='qcow2'/>
      <source file='/path/to/new'/>
    </disk>
    <disk name='vdb' snapshot='no'/>
  </disks>
  <domain>
    <name>fedora</name>
    <uuid>93a5c045-6457-2c09-e56c-927cdf34e178</uuid>
    <memory>1048576</memory>
    ...
    <devices>
      <disk type='file' device='disk'>
        <driver name='qemu' type='raw'/>
        <source file='/path/to/old'/>
        <target dev='vda' bus='virtio'/>
      </disk>
      <disk type='file' device='disk' snapshot='external'>
        <driver name='qemu' type='raw'/>
        <source file='/path/to/old2'/>
        <target dev='vdb' bus='virtio'/>
      </disk>
      ...
    </devices>
  </domain>
</domainsnapshot>
```

With that snapshot created, `/path/to/old` is the read-only backing file to the new active file `/path/to/new`. The `<domain>` element within the snapshot `xml` records the state of the domain just before the snapshot; a call to `getXMLDesc()` will show that the domain has been changed to reflect the snapshot (Listing A-49).

Listing A-49. Sample XML from the getXMLDesc() Method

```
<domain>
  <name>fedora</name>
  <uuid>93a5c045-6457-2c09-e56c-927cdf34e178</uuid>
  <memory>1048576</memory>
  ...
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/path/to/new'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <disk type='file' device='disk' snapshot='external'>
      <driver name='qemu' type='raw'/>
      <source file='/path/to/old2'/>
      <target dev='vdb' bus='virtio'/>
    </disk>
    ...
  </devices>
</domain>
```

The Domain Capabilities XML Format

Sometimes, when a new domain is to be created, it may become handy to know the capabilities of the hypervisor so the correct combination of devices and drivers is used. For example, when a management application is considering the mode for a host device's passthrough, there are several options depending not only on host but on the hypervisor in question. If the hypervisor is QEMU, then it needs to be more recent to support VFIO, while legacy KVM is achievable just fine with older qemus.

The main difference between the connection method `getCapabilities` and the emulator capabilities API is that the former one focuses more on the host capabilities (e.g., NUMA topology, security models in effect, etc.), while the latter one specializes on the hypervisor capabilities.

While the Driver Capabilities section provides the host capabilities (e.g., NUMA topology, security models in effect, etc.), the "Domain Capabilities" section provides the hypervisor-specific capabilities for management applications to query and make decisions regarding what to utilize.

The Domain Capabilities section can provide information such as the correct combination of devices and drivers that are supported. Knowing which host and hypervisor-specific options are available or supported will allow the management application to choose an appropriate mode for a passthrough host device as well as which adapter to utilize.

Element and Attribute Overview

Listing A-50 shows the new query interface that was added to the `virConnect` APIs to retrieve the XML listing of the set of domain capabilities (since 1.2.7).

Listing A-50. The Method for Retieving Domain Capabilities

```
getDomainCapabilities()
```

The root element that the emulator capability XML document starts with has the name `<domainCapabilities>`. It contains at least four direct child elements (Listing A-51).

Listing A-51. A Sample Domain Capabilities XML Output

```
<domainCapabilities>
  <path>/usr/bin/qemu-system-x86_64</path>
  <domain>kvm</domain>
  <machine>pc-i440fx-2.1</machine>
  <arch>x86_64</arch>
  ...
</domainCapabilities>
```

CPU Allocation

Before any device's capability occurs, there might be information on domain-wide capabilities, e.g., virtual CPUs (Listing A-52).

Listing A-52. Retrieving CPU Capabilities

```
<domainCapabilities>
  ...
  <vcpu max='255'/>
  ...
</domainCapabilities>
```

BIOS Bootloader

Sometimes users might want to tweak some BIOS knobs or use UEFI. For cases like that, the `os` element exposes what values can be passed to its children (Listing A-53) .

Listing A-53. Retrieving BIOS or UEFI Capabilities from a Guest Domain

```
<domainCapabilities>
  ...
  <os supported='yes'>
    <loader supported='yes'>
      <value>/usr/share/OVMF/OVMF_CODE.fd</value>
      <enum name='type'>
        <value>rom</value>
        <value>pflash</value>
      </enum>
      <enum name='readonly'>
        <value>yes</value>
        <value>no</value>
      </enum>
    </loader>
  </os>
  ...
<domainCapabilities>
```

CPU Configuration

Each CPU mode understood by libvirt is described with a mode element, which tells whether the particular mode is supported and provides (when applicable) more details about it (Listing A-54).

Listing A-54. Retrieving CPU Configuration of a Guest Domain

```
<domainCapabilities>
  ...
  <cpu>
    <mode name='host-passthrough' supported='yes'/>
    <mode name='host-model' supported='yes'>
      <model fallback='allow'>Broadwell</model>
      <vendor>Intel</vendor>
      <feature policy='disable' name='aes'/>
      <feature policy='require' name='vmx'/>
    </mode>
    <mode name='custom' supported='yes'>
      <model usable='no'>Broadwell</model>
      <model usable='yes'>Broadwell-noTSX</model>
      <model usable='no'>Haswell</model>
      ...
    </mode>
  </cpu>
  ...
<domainCapabilities>
```

IO Threads

The <iothread> elements indicates whether I/O threads are supported (Listing A-55).

Listing A-55. Retrieving iothread Capabilities for a Guest Domain

```
<domainCapabilities>
  ...
  <iothread supported='yes'/>
  ...
<domainCapabilities>
```

Reported capabilities are expressed as an enumerated list of available options for each element or attribute. For example, the <disk/> element has an attribute device that can support the values <disk>, <cdrom>, <floppy>, and <lun>.

Hard Drives, Floppy Disks, and CD-ROMs

Disk capabilities are exposed under the disk element, as shown in Listing A-56.

Listing A-56. Retrieving Drive Capabilities from a Guest Domain

```
<domainCapabilities>
  ...
  <devices>
    <disk supported='yes'>
      <enum name='diskDevice'>
        <value>disk</value>
        <value>cdrom</value>
        <value>floppy</value>
        <value>lun</value>
      </enum>
      <enum name='bus'>
        <value>ide</value>
        <value>fdc</value>
        <value>scsi</value>
        <value>virtio</value>
        <value>xen</value>
        <value>usb</value>
        <value>sata</value>
        <value>sd</value>
      </enum>
    </disk>
    ...
  </devices><domainCapabilities>
```

Graphical Framebuffers

Graphics device capabilities are exposed under the `graphics` element (Listing A-57).

Listing A-57. Retriving Graphical Device Capabilities from a Guest Domain

```
<domainCapabilities>
  ...
  <devices>
    <graphics supported='yes'>
      <enum name='type'>
        <value>sdl</value>
        <value>vnc</value>
        <value>spice</value>
      </enum>
    </graphics>
    ...
  </devices>  </devices><domainCapabilities>
```

Video Device

Video device capabilities are exposed under the `video` element (Listing A-58).

Listing A-58. Retrieving Video Capabilities from a Guest Domain

```
<domainCapabilities>
  ...
  <devices>
    <video supported='yes'>
      <enum name='modelType'>
        <value>vga</value>
        <value>cirrus</value>
        <value>vmvga</value>
        <value>qxl</value>
        <value>virtio</value>
      </enum>
    </video>
    ...
  </devices>
<domainCapabilities>
```

Host Device Assignment

Some host devices can be passed through to a guest (e.g., USB, PCI, and SCSI), but only if the code in Listing A-59 is enabled.

Listing A-59. Retrieving Host Assignments to the Guest Domain

```
<domainCapabilities>
  ...
  <devices>
    <hostdev supported='yes'>
      <enum name='mode'>
        <value>subsystem</value>
        <value>capabilities</value>
      </enum>
      <enum name='startupPolicy'>
        <value>default</value>
        <value>mandatory</value>
        <value>requisite</value>
        <value>optional</value>
      </enum>
      <enum name='subsysType'>
        <value>usb</value>
        <value>pci</value>
        <value>scsi</value>
      </enum>
      <enum name='capsType'>
        <value>storage</value>
        <value>misc</value>
        <value>net</value>
      </enum>
      <enum name='pciBackend'>
        <value>default</value>
        <value>kvm</value>
        <value>vfio</value>
        <value>xen</value>
      </enum>
    </hostdev>
  </devices>
<domainCapabilities>
```

Features

One more set of XML elements describes the supported features and their capabilities. All `<features>` elements occur as children of the main `features` element (Listing A-60).

Listing A-60. Retrieving Supported Features from the Guest Domain

```
<domainCapabilities>
  ...
  <features>
    <gic supported='yes'>
      <enum name='version'>
        <value>2</value>
        <value>3</value>
      </enum>
    </gic>
    <vmcoreinfo supported='yes'/>
    <genid supported='yes'/>
    <sev>
      <cbitpos>47</cbitpos>
      <reduced-phys-bits>1</reduced-phys-bits>
    </sev>
  </features>
<domainCapabilities>
```

Reported capabilities are expressed as an enumerated list of possible values for each element or attribute. For example, the `<gic>` element has an attribute version that can support the value 2 or 3.

For information about the purpose of each feature, see the relevant section in the domain XML documentation.

Network Filters

This section provides an introduction to libvirt's network filters, including their goals, concepts, and XML format.

Goals and Background

The goal of the network filtering XML is to enable administrators of a virtualized system to configure and enforce network traffic filtering rules on virtual machines and manage the parameters of network traffic that virtual machines are allowed to send or receive. The network traffic filtering rules are applied on the host when a virtual machine is started. Since the filtering rules cannot be circumvented from within the virtual machine, it makes them mandatory from the point of view of a virtual machine user.

The network filter subsystem allows each virtual machine's network traffic filtering rules to be configured individually on a per-interface basis. The rules are applied on the host when the virtual machine is started and can be modified while the virtual machine is running. The latter can be achieved by modifying the XML description of a network filter.

Multiple virtual machines can make use of the same generic network filter. When such a filter is modified, the network traffic filtering rules of all running virtual machines that reference this filter are updated.

Network filtering support has been available since version 0.8.1 (QEMU, KVM).

Concepts

The network traffic filtering subsystem enables configuration of network traffic filtering rules on individual network interfaces that are configured for certain types of network configurations. Supported network types are as follows:

- Network
- Ethernet (must be used in bridging mode)
- Bridge

The interface XML is used to reference a top-level filter. In Listing A-61, the interface description references the filter `clean-traffic`.

Listing A-61. Specifying Network Filters

```
...
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e'/>
    <filterref filter='clean-traffic'/>
  </interface>
</devices>
...
```

Network filters are written in XML and may either contain references to other filters, contain rules for traffic filtering, or hold a combination of both. The referenced filter `clean-traffic` is a filter that only contains references to other filters and no actual filtering rules. Since references to other filters can be used, a tree of filters can be built. The `clean-traffic` filter can be viewed using the command `virsh nwfilter-dumpxml clean-traffic`.

As previously mentioned, a single network filter can be referenced by multiple virtual machines. Since interfaces will typically have individual parameters associated with their respective traffic filtering rules, the rules described in a filter XML can be parameterized with variables. In this case, the variable name is used in the filter XML, and the name and value are provided at the place where the filter is referenced. In Listing A-62, the interface description has been extended with the parameter IP and a dotted IP address as the value.

Listing A-62. Specifying IP-Specific Network Filters

```
...
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e'/>
    <filterref filter='clean-traffic'>
      <parameter name='IP' value='10.0.0.1'/>
    </filterref>
  </interface>
</devices>
...
```

In this particular example, the `clean-traffic` network traffic filter will be instantiated with the IP address parameter 10.0.0.1 and enforce that the traffic from this interface will always be using 10.0.0.1 as the source IP address, which is one of the purposes of this particular filter.

Filtering Chains

Filtering rules are organized in filter chains. These chains can be thought of as having a tree structure with packet filtering rules as entries in individual chains (branches).

Packets start their filter evaluation in the root chain and can then continue their evaluation in other chains, return from those chains into the root chain, or be dropped or accepted by a filtering rule in one of the traversed chains.

libvirt's network filtering system automatically creates individual root chains for every virtual machine's network interface on which the user chooses to activate traffic filtering. The user may write filtering rules that either are directly instantiated in the root chain or may create protocol-specific filtering chains for efficient evaluation of protocol-specific rules. The following chains exist:

- `root`
- `mac` (since 0.9.8)
- `stp` (Spanning Tree Protocol) (since 0.9.8)
- `vlan` (802.1Q) (since 0.9.8)
- `arp`, `rarp`
- `ipv4`
- `ipv6`

Since 0.9.8, multiple chains evaluating the `mac`, `stp`, `vlan`, `arp`, `rarp`, `ipv4`, `and ipv6` protocols can be created using the protocol name only as a prefix in the chain's name. This, for example, allows chains with the names arp-xyz or arp-test to be specified and have the ARP protocol packets evaluated in those chains.

The filter in Listing A-63 shows an example of filtering ARP traffic in the arp chain.

Listing A-63. Specifying Filtering ARP Traffic in the arp Chain

```
<filter name='no-arp-spoofing' chain='arp' priority='-500'>
  <uuid>f88f1932-debf-4aa1-9fbe-f10d3aa4bc95</uuid>
  <rule action='drop' direction='out' priority='300'>
    <mac match='no' srcmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='out' priority='350'>
    <arp match='no' arpsrcmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='out' priority='400'>
    <arp match='no' arpsrcipaddr='$IP'/>
  </rule>
  <rule action='drop' direction='in' priority='450'>
    <arp opcode='Reply'/>
    <arp match='no' arpdstmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='in' priority='500'>
    <arp match='no' arpdstipaddr='$IP'/>
  </rule>
  <rule action='accept' direction='inout' priority='600'>
    <arp opcode='Request'/>
  </rule>
  <rule action='accept' direction='inout' priority='650'>
    <arp opcode='Reply'/>
  </rule>
  <rule action='drop' direction='inout' priority='1000'/>
</filter>
```

The consequence of putting ARP-specific rules in the `arp` chain, rather than, for example, in the `root` chain, is that packets for any other protocol than ARP do not need to be evaluated by ARP-specific rules. This improves the efficiency of the traffic filtering. However, one must then pay attention to only put filtering rules for the given protocol into the chain since any other rules will not be evaluated; in other words, an IPv4 rule will not be evaluated in the `arp` chain since no IPv4 protocol packets will traverse the `arp` chain.

Filtering Chain Priorities

All chains are connected to the root chain. The order in which those chains are accessed is influenced by the priority of the chain. Table A-1 shows the chains that can be assigned a priority and their default priorities.

Table A-1. *Filtering Chain Priorities*

Chain (Prefix)	Default Priority
`stp`	-810
`mac`	-800
`stp`	-810
`vlan`	-750
`ipv4`	-700
`ipv6`	-600
`arp`	-500
`rarp`	-400

A chain with a lower-priority value is accessed before one with a higher value. Since 0.9.8, the previously listed chains can be assigned custom priorities by writing a value in the range `[-1000, 1000]` into the priority (XML) attribute in the filter node. The previous example filter shows the default priority of -500 for `arp` chains.

Usage of Variables in Filters

Two variables names have so far been reserved for usage by the network traffic filtering subsystem: `MAC` and `IP`.

`MAC` is the MAC address of the network interface. A filtering rule that references this variable will automatically be instantiated with the MAC address of the interface. This works without the user having to explicitly provide the `MAC` parameter. Even though it is possible to specify the `MAC` parameter similar to the `IP` parameter shown previously, it is discouraged since libvirt knows what MAC address an interface will be using.

The parameter `IP` represents the IP address that the operating system inside the virtual machine is expected to use on the given interface. The `IP` parameter is special in so far as the libvirt daemon will try to determine the IP address (and thus the `IP` parameter's value) that is being used on an interface if the parameter is not explicitly provided but referenced. For current limitations on IP address detection, consult the Chapter 7.

The previously shown network filter `no-arp-spoofing` is an example of a network filter with XML referencing the `MAC` and `IP` variables.

Note that referenced variables are always prefixed with the dollar (`$`) sign. The format of the value of a variable must be of the type expected by the filter attribute in the XML. In the previous example, the `IP` parameter must hold a dotted IP address in decimal number format. Failure to provide the correct value type will result in the filter not being instantiatable and will prevent a virtual machine from starting or the interface from attaching when hot plugging is used. The types that are expected for each XML attribute are shown in Listing A-64.

Since version 0.9.8, variables can contain lists of elements; for example, the variable `IP` can contain multiple IP addresses that are valid on a particular interface. Listing A-64 shows the notation for providing multiple elements for the `IP` variable.

Listing A-64. Specifying an IP Filter for Specific Network Addresses

```
...
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e'/>
    <filterref filter='clean-traffic'>
      <parameter name='IP' value='10.0.0.1'/>
      <parameter name='IP' value='10.0.0.2'/>
      <parameter name='IP' value='10.0.0.3'/>
    </filterref>
  </interface>
</devices>
...
```

This then allows filters to enable multiple IP addresses per interface. Therefore, with the list of IP addresses shown previously, the rule in Listing A-65 will create three individual filtering rules, one for each IP address.

Listing A-65. Specifying an IP Filter for Multiple IP Addresses

```
...
<rule action='accept' direction='in' priority='500'>
  <tcp srpipaddr='$IP'/>
</rule>
...
```

Since version 0.9.10, it is possible to access individual elements of a variable holding a list of elements. A filtering rule like that in Listing A-66 accesses the second element of the variable `DSTPORTS`.

Listing A-66. Specifying an IP Filter for Specific Ports

```
...
<rule action='accept' direction='in' priority='500'>
  <udp dstportstart='$DSTPORTS[1]'/>
</rule>
...
```

Since 0.9.10, it is possible to create filtering rules that instantiate all combinations of rules from different lists using the notation of `$VARIABLE[@<iterator ID>]`. The rule in Listing A-67 allows a virtual machine to receive traffic on a set of ports, which are specified in `DSTPORTS`, from the set of source IP address specified in `SRCIPADDRESSES`. The rule generates all combinations of elements of the variable `DSTPORT` with those of `SRCIPADDRESSES` by using two independent iterators to access their elements.

Listing A-67. Specify a Filter for a Range of Ports

```
...
<rule action='accept' direction='in' priority='500'>
  <ip srcipaddr='$SRCIPADDRESSES[@1]'
dstportstart='$DSTPORTS[@2]'/>
</rule>
...
```

Listing A-68 assigns concrete values to `SRCIPADDRESSES` and `DSTPORTS`.

Listing A-68. Specifying a Filter for Concrete Values

```
SRCIPADDRESSES = [ 10.0.0.1, 11.1.2.3 ]
DSTPORTS = [ 80, 8080 ]
```

Accessing the variables using `$SRCIPADDRESSES[@1]` and `$DSTPORTS[@2]` would then result in all combinations of addresses and ports being created (Listing A-69).

Listing A-69. Result of Specifying a Filter for a Range of Concrete Addresses

```
10.0.0.1, 80
10.0.0.1, 8080
11.1.2.3, 80
11.1.2.3, 8080
```

Accessing the same variables using a single iterator, for example by using the notation `$SRCIPADDRESSES[@1]` and `$DSTPORTS[@1]`, would result in parallel access to both lists and result in the combinations shown in Listing A-70.

Listing A-70. Result of Specifying a Filter for a Range of Concrete Addresses

```
10.0.0.1, 80
11.1.2.3, 8080
```

Further, the notation of `$VARIABLE` is shorthand for `$VARIABLE[@0]`. The former notation always assumes an iterator with an ID of 0.

Automatic IP Address Detection

The detection of IP addresses used on a virtual machine's interface is automatically activated if the variable `IP` is referenced but no value has been assigned to it. Since 0.9.13, the variable `CTRL_IP_LEARNING` can be used to specify the IP address learning method to use. Valid values are `any`, `dhcp`, `and none`.

The value any means that libvirt may use any packet to determine the address in use by a virtual machine, which is the default behavior if the variable CTRL_IP_LEARNING is not set. This method will detect only a single IP address on an interface. Once a VM's IP address has been detected, its IP network traffic will be locked to that address, if for example IP address spoofing is prevented by one of its filters. In that case, the user of the VM will not be able to change the IP address on the interface inside the VM, which would be considered IP address spoofing. When a VM is migrated to another host or resumed after a suspend operation, the first packet sent by the VM will again determine the IP address it can use on a particular interface.

A value of dhcp specifies that libvirt should honor only DHCP server-assigned addresses with valid leases. This method supports the detection and usage of multiple IP addresses per interface. When a VM is resumed after a suspend operation, still-valid IP address leases are applied to its filters. Otherwise, the VM is expected to again use DHCP to obtain new IP addresses. The migration of a VM to another physical host requires that the VM again runs the DHCP protocol.

Using CTRL_IP_LEARNING=dhcp (DHCP snooping) provides additional antispoofing security, especially when combined with a filter allowing only trusted DHCP servers to assign addresses. To enable this, set the variable DHCPSERVER to the IP address of a valid DHCP server and provide filters that use this variable to filter incoming DHCP responses.

When DHCP snooping is enabled and the DHCP lease expires, the VM will no longer be able to use the IP address until it acquires a new, valid lease from a DHCP server. If the VM is migrated, it must get a new valid DHCP lease to use an IP address (e.g., by bringing the VM interface down and up again).

Note that automatic DHCP detection listens to the DHCP traffic the VM exchanges with the DHCP server of the infrastructure. To avoid denial-of-service attacks on libvirt, the evaluation of those packets is rate-limited, meaning that a VM sending an excessive number of DHCP packets per second on an interface will not have all of those packets evaluated, and thus filters may not get adapted. Normal DHCP client behavior is assumed to send a low number of DHCP packets per second. Further, it is important to set up appropriate filters on all VMs in the infrastructure to avoid them being able to send DHCP packets. Therefore, either VMs must be prevented from sending UDP and TCP traffic from port 67 to port 68 or the `DHCPSERVER` variable should be used on all VMs to restrict DHCP server messages to be allowed to originate only from trusted DHCP servers. At the same time, antispoofing prevention must be enabled on all VMs in the subnet.

If `CTRL_IP_LEARNING` is set to `none`, libvirt does not do IP address learning, and referencing `IP` without assigning it an explicit value is an error.

The XML in Listing A-71 provides an example for the activation of IP address learning using the DHCP snooping method.

Listing A-71. Activating a Specific IP Address

```
<interface type='bridge'>
  <source bridge='virbr0'/>
  <filterref filter='clean-traffic'>
    <parameter name='CTRL_IP_LEARNING' value='dhcp'/>
  </filterref>
</interface>
```

Reserved Variables

Table A-2 lists reserved variables in use by libvirt.

Table A-2. *Libvirt Reserved Variables*

Variable Name	Semantics
`MAC`	The MAC address of the interface
`IP`	The list of IP addresses in use by an interface
`IPV6`	Not currently implemented: the list of IPV6 addresses in use by an interface
`DHCPSERVER`	The list of IP addresses of trusted DHCP servers
`DHCPSERVER6`	Not currently implemented: the list of IPv6 addresses of trusted DHCP servers
`CTRL_IP_LEARNING`	The choice of the IP address detection mode

Element and Attribute Overview

The root element required for all network filters is named `filter` with two possible attributes. The `name` attribute provides a unique name of the given filter. The `chain` attribute is optional but allows certain filters to be better organized for more efficient processing by the firewall subsystem of the underlying host. Currently the system supports only the chains `root`, `ipv4`, `ipv6`, `arp`, and `rarp`.

References to Other Filters

Any filter may hold references to other filters. Individual filters may be referenced multiple times in a filter tree, but references between filters must not introduce loops (directed acyclic graph).

Listing A-72 shows the XML of the `clean-traffic` network filter referencing several other filters.

Listing A-72. Specifying a clean-traffic Filter Referencing Other Filters

```
<filter name='clean-traffic'>
  <uuid>6ef53069-ba34-94a0-d33d-17751b9b8cb1</uuid>
  <filterref filter='no-mac-spoofing'/>
  <filterref filter='no-ip-spoofing'/>
  <filterref filter='allow-incoming-ipv4'/>
  <filterref filter='no-arp-spoofing'/>
  <filterref filter='no-other-l2-traffic'/>
  <filterref filter='qemu-announce-self'/>
</filter>
```

To reference another filter, the XML node `filterref` needs to be provided inside a filter node. This node must have the attribute filter whose value contains the name of the filter to be referenced.

New network filters can be defined at any time and may contain references to network filters that are not known to libvirt yet. However, once a virtual machine is started or a network interface referencing a filter is to be hot plugged, all network filters in the filter tree must be available. Otherwise, the virtual machine will not start or the network interface cannot be attached.

Filter Rules

Listing A-73 shows a simple example of a network traffic filter implementing a rule to drop traffic if the IP address (provided through the value of the variable `IP`) in an outgoing IP packet is not the expected one, thus preventing IP address spoofing by the VM.

Listing A-73. Using Filter Rules

```
<filter name='no-ip-spoofing' chain='ipv4'>
  <uuid>fce8ae33-e69e-83bf-262e-30786c1f8072</uuid>
  <rule action='drop' direction='out' priority='500'>
    <ip match='no' srcipaddr='$IP'/>
  </rule>
</filter>
```

A traffic filtering rule starts with the rule node. This node may contain up to three attributes.

- `action`: Mandatory; must be `drop` (matching the rule silently discards the packet with no further analysis), `reject` (matching the rule generates an ICMP reject message with no further analysis) since 0.9.0, `accept` (matching the rule accepts the packet with no further analysis), `return` (matching the rule passes this filter but returns control to the calling filter for further analysis) since 0.9.7, or `continue` (matching the rule goes on to the next rule for further analysis) since 0.9.7.
- `direction`: Mandatory; must be either `in`, `out`, or `inout` if the rule is for incoming, outgoing, or incoming and outgoing traffic.
- `priority`: Optional; the priority of the rule controls the order in which the rule will be instantiated relative to other rules. Rules with lower value will be instantiated before rules with higher values. Valid values are in the range of 0 to 1000. Since 0.9.8, this has been extended to cover the range of -1000 to 1000. If this attribute is not provided, priority 500 will automatically be assigned. Note that filtering rules in the root chain are

sorted with filters connected to the root chain following their priorities. This allows you to interleave filtering rules with access to filter chains. (See also the "Filtering Chain Priorities" section.)

- `statematch`: Optional; possible values are `0` or `false` to turn the underlying connection state matching off; the default is `true`. Also read the "Network Filters" section.

The previous example indicates that the traffic of type `ip` will be associated with the chain `ipv4`, and the rule will have priority 500. If for example another filter is referenced whose traffic of type `ip` is also associated with the chain `ipv4`, then that filter's rules will be ordered relative to the priority 500 of the shown rule.

A rule may contain a single rule for the filtering of traffic. The previous example shows that traffic of type `ip` is to be filtered.

Supported Protocols

The following sections list the protocols that are supported by the network filtering subsystem. The type of traffic a rule is supposed to filter on is provided in the rule node as a nested node. Depending on the traffic type that a rule is filtering, the attributes are different. The previous example showed the single attribute `srcipaddr` that is valid inside the IP traffic filtering node. The following sections show what attributes are valid and what type of data they are expecting. The following datatypes are available:

- `UINT8`: 8-bit integer; range 0–255.
- `UINT16`: 16-bit integer; range 0–65535.
- `MAC_ADDR`: MAC address in dotted decimal format, i.e., 00:11:22:33:44:55.
- `MAC_MASK`: MAC address mask in MAC address format, i.e., FF:FF:FF:FC:00:00.

- `IP_ADDR`: IP address in dotted decimal format, i.e., 10.1.2.3.
- `IP_MASK`: IP address mask in either dotted decimal format (255.255.248.0) or CIDR mask (0–32).
- `IPV6_ADDR`: IPv6 address in numbers format, i.e., FFFF::1.
- `IPV6_MASK`: IPv6 mask in numbers format (FFFF:FFFF:FC00::) or CIDR mask (0–128).
- `STRING`: A string.
- `BOOLEAN`: `true`, `yes`, `1` or `false`, `no`, `0'5`.
- `IPSETFLAGS`: The source and destination flags of the `ipset` described by up to six `src` or `dst` elements selecting features from either the source or the destination part of the packet header. Here's an example: `src`,`src`,`dst`. The number of `selectors` to provide here depends on the type of `ipset` that is referenced.

Every attribute except for those of type `IP_MASK` or `IPV6_MASK` can be negated using the `match` attribute with the value `no`. Multiple negated attributes may be grouped together. The XML fragment in Listing A-74 shows such an example using abstract attributes.

Listing A-74. Negating Attributes

```
...
<rule action='drop' direction='in'>
  <protocol match='no' attribute1='value1'
attribute2='value2'/>
  <protocol attribute3='value3'/>
</rule>
...
```

Rules perform a logical AND evaluation on all values of the given protocol attributes. Thus, if a single attribute's value does not match the one given in the rule, the whole rule will be skipped during evaluation. Therefore, in the previous example, incoming traffic will be dropped only if the protocol property `attribute1` does not match `value1` and the protocol property `attribute2` does not match `value2` and the protocol property `attribute3` matches `value3`.

MAC (Ethernet)

Protocol ID: `mac` (Table A-3)

Table A-3. *mac Variables and Their Expected Values*

Variable Name	Datatype	Semantics
`srcmacaddr`	`MAC_ADDR`	MAC address of sender
`srcmacmask`	`MAC_MASK`	Mask applied to MAC address of sender
`dstmacaddr`	`MAC_ADDR`	MAC address of destination
`dstmacmask`	`MAC_MASK`	Mask applied to MAC address of destination
`protocolid`	`MAC_ADDR`	MAC address of sender
`MAC`	`UINT16 (0x600-0xffff), STRING`	`UINT16 (0x600-0xffff), STRING`
`comment` (since 0.8.5)	`STRING`	Text with a maximum of 256 characters

Note: Rules of this type should go into the `root` chain.

Listing A-75 shows that the valid strings for `protocolid` are `arp`, `rarp`, `ipv4`, and `ipv6`.

Listing A-75. Valid Strings for protocolid

```
...
<rule action='drop' direction='in'>
  <protocol match='no' attribute1='value1'
attribute2='value2'/>
  <protocol attribute3='value3'/>
</rule>
...
```

VLAN (802.1Q) (Since 0.9.8)

Protocol ID: vlan (Table A-4)

Table A-4. *vlan Variables and Their Expected Values*

Variable Name	Datatype	Semantics
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
vlanid	UINT16 (0x0-0xfff, 0–4095)	VLAD ID
encap-protocol	UINT16 (0x03c-0xfff), String	Encapsulated layer 3 protocol ID
comment (since 0.8.5)	STRING	Text with a maximum of 256 characters

Note: Rules of this type should go into either the `vlan` or `root` chain. Valid strings for `encap-protocol` are `arp`, `ipv4`, and `ipv6`.

STP (Spanning Tree Protocol) (Since 0.9.8)

Protocol ID: `stp` (Table A-5)

Table A-5. *stp Variables and Their Expected Values*

Variable Name	Datatype	Semantics
`srcmacaddr`	`MAC_ADDR`	MAC address of sender
`srcmacmask`	`MAC_MASK`	Mask applied to MAC address of sender
`type`	`UINT8`	Bridge Protocol Data Unit (BPDU) type
`flags`	`UINT8`	BPDU flag
`root-priorityvlanid`	`UINT16`	Root priority (range start)
`root-priority-hi`	`UINT16`	Root priority (range end)
`root-address`	`MAC_ADDRESS`	Root MAC address
`root-address-mask`	`MAC_MASK`	Root MAC address mask
`root-cost`	`UINT32`	Root path cost (range start)
`root-cost-hi`	`UINT32`	Root path cost range end
`sender-priority`	`UINT16`	Sender priority (range start)
`sender-priority-hi`	`UINT16`	Sender priority range end
`sender-address`	`MAC_ADDRESS6`	BPDU sender MAC address
`sender-address-mask`	`MAC_MASK`	BPDU sender MAC address mask
`port`	`UIMT16`	Port identifier (range start)

(*continued*)

Table A-5. (*continued*)

Variable Name	Datatype	Semantics
`port-hi`	`UIMT16`	Port identifier range end)
`msg-age`	`UIMT16`	Message age timer (range start)
`msg-age-hi`	`UIMT16`	Message age timer range end
`max-age`	`UIMT16`	Maximum age timer (range start)
`max-age-hi`	`UIMT16`	Maximum age timer range end
`hello-time`	`UIMT16`	Hello time timer (range start)
`hello-time-hi`	`UIMT16`	Hello time timer range end
`forward-delay`	`UIMT16`	Forward delay (range start)
`forward-delay`	`UIMT16`	Forward delay range end
`forward-delay-hi`	`UIMT16`	Forward delay range end
`comment`	`STRING`	Text with maximum of 256 characters

Note: Rules of this type should go into either the `vlan` or the `root` chain.

ARP/RARP

Protocol ID: `apr` or `rarp` (Table A-6)

***Table A-6.** apr or rarp Variables and Their Expected Values*

Variable Name	Datatype	Semantics
`srcmacaddr`	`MAC_ADDR`	MAC address of sender
`srcmacmask`	`MAC_MASK`	Mask applied to MAC address of sender
`dstmacaddr`	`MAC_ADDR`	MAC address of destination
`dstmacmask`	`MAC_MASK`	Mask applied to MAC address of destination
`hwtype`	`UINT16`	Hardware type
`protocoltype`	`UINT16`	Protocol type
`opcode`	`UINT16, STRING`	Opcode
`arpsrcmacaddr`	`MAC_ADDR`	Source MAC address in ARP/RARP packet
`arpdstmacaddr`	`MAC_ADDR`	Destination MAC address in ARP/RARP packet
`arpsrcipaddr`	`IP_ADDR`	Source IP address in ARP/RARP packet
`arpsrcipmask` (Since 1.2.3)	`IP_MASK`	Source IP mask
`arpdstipaddr`	`IP_ADDR`	Destination IP address in ARP/RARP packet
`arpdstipmask` (Since 1.2.3)	`IP_MASK`	Destination IP mask
`comment` (Since 0.8.5)	`STRING`	Text with a maximum 256 characters
`gratuitous` (Since 0.9.2)	`BOOLEAN`	Boolean indicating whether to check for gratuitous ARP packet

Note: Rules of this type should go into either the root or the arp/rarp chain.

Valid strings for the Opcode field are Request, Reply, Request_Reverse, Reply_Reverse, DRARP_Request, DRARP_Reply, DRARP_Error, InARP_Request, and ARP_NAK.

IPv4

Protocol ID: ip (Table A-7)

Table A-7. *ip Variables and Their Expected Values*

Variable Name	Datatype	Semantics
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
protocol	UINT8, STRING	Layer 4 protocol identifier
srcportstart	UINT16	Start of range of valid source ports; requires protocol
srcportend	UINT16	End of range of valid source ports; requires protocol

(*continued*)

Table A-7. *(continued)*

Variable Name	Datatype	Semantics
dstportstart	UINT16	Start of range of valid destination ports; requires protocol
dstportend	UINT16	End of range of valid destination ports; requires protocol
dscp	UINT8 (0x0-0x3f, 0 - 63)	Differentiated Services Code Point
comment (Since 0.8.5)	STRING	Text with a maximum of 256 characters

Note: Rules of this type should go into either the root or the ipv4 chain.

Valid strings for protocol are tcp, udp, udplite, esp, ah, icmp, igmp, and sctp.

IPv6

Protocol ID: ipv6 (Table A-8)

Table A-8. *ipv6 Variables and Their Expected Values*

Variable Name	Datatype	Semantics
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
srcipaddr	IPV6_ADDR	Source IP address

(continued)

Table A-8. (*continued*)

Variable Name	Datatype	Semantics
`srcipmask`	`IPV6_MASK`	Mask applied to source IP address
`dstipaddr`	`IPV6_ADDR`	Destination IP address
`dstipmask`	`IPV6_MASK`	Mask applied to destination IP address
`protocol`	`UINT8`	Layer 4 protocol identifier
`srcportstart`	`UINT16`	Start of range of valid source ports; requires protocol
`srcportend`	`UINT16`	End of range of valid source ports; requires protocol
`dstportstart`	`UINT16`	Start of range of valid destination ports; requires protocol
`dstportend`	`UINT16`	End of range of valid destination ports; requires protocol
`type` (Since 1.2.12)	`UINT8`	Differentiated Services Code Point
`typeend` (Since 1.2.12)	`UINT8`	ICMPv6 type end of range; requires protocol to be set to icmpv6
`code` (Since 1.2.12)	`UINT8`	ICMPv6 code; requires protocol to be set to icmpv6
`codeend` (Since 1.2.12)	`UINT8`	ICMPv6 code end of range; requires protocol to be set to icmpv6
`comment` (Since 0.8.5)	`STRING`	Text with a maximum of 256 characters

Note: Rules of this type should go into either the `root` or the `ipv6` chain.

Valid strings for `protocol` are `tcp`, `udp`, `udplite`, `esp`, `ah`, `icmpv6`, and `sctp`.

TCP/UDP/SCTP

Protocol ID: tcp, udp, sctp (Table A-9)

Table A-9. *tcp, udp, sctp Variables and Their Expected Values*

Variable Name	Datatype	Semantics
srcmacaddr	MAC_ADDR	MAC address of sender
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	Start of range of source IP address
srcipto	IP_ADDR	End of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
srcportstart	UINT16	Start of range of valid source ports
srcportend	UINT16	End of range of valid source port
dstportstart	UINT16	Start of range of valid destination ports
dstportend	UINT16	End of range of valid destination ports
dscp	UINT8 (0x0-0x3f, 0–63)	Differentiated Services Code Point
comment (since 0.8.5)	STRING	Text with a maximum of 256 characters
state (since 0.8.5)	STRING	Comma-separated list of NEW, ESTABLISHED, RELATED, INVALID, or NONE

(*continued*)

Table A-9. *(continued)*

Variable Name	Datatype	Semantics
flags (since 0.9.1)	STRING	TCP-only: Format of mask/flags with mask and flags, each being a comma-separated list of SYN, ACK, URG, PSH, FIN, RST, NONE, or ALL6
ipset (since 0.9.13)	STRING	The name of an IPSet managed outside of libvirt
ipsetflags (since 0.9.13)	IPSETFLAGS	Flags for the IPSet; requires ipset attribute

Note: The chain parameter is ignored for this type of traffic and should be either omitted or set to the root chain.

ICMP

Protocol ID: icmp (Table A-10)

Table A-10. *icmp Variables and Their Expected Values*

Variable Name	Datatype	Semantics
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
srcipaddr	IP_ADDR	Source IP address

(*continued*)

Table A-10. (*continued*)

Variable Name	Datatype	Semantics
`srcipmask`	`IP_MASK`	Mask applied to source IP address
`dstipaddr`	`IP_ADDR`	Destination IP address
`dstipmask`	`IP_MASK`	Mask applied to destination IP addresss
`srcipfromo`	`IP_ADDR`	Start of range of source IP address
`srcipto`	`IP_ADDR`	End of range of source IP address
`dstipfrom`	`IP_ADDR`	Start of range of destination IP address
`dstipto`	`IP_ADDR`	End of range of destination IP address
`type`	`UINT16`	ICMP type
`code`	UINT16	ICMP code
`dscp`	UINT8 (0x0-0x3f, 0 - 63)	Differentiated Services Code Point
`comment` (since 0.8.5)	STRING	Text with a maximum of 256 characters
`state` (since 0.8.5)	`STRING`	Comma-separated list of `NEW`, `ESTABLISHED`, `RELATED`, `INVALID`, or `NONE`
`ipset` (since 0.9.13)	`STRING`	The name of an IPSet managed outside of libvirt
`ipsetflags` (since 0.9.13)	`IPSETFLAGS`	Flags for the IPSet; requires `ipset` attribute

Note: The chain parameter is ignored for this type of traffic and should be either omitted or set to the `root` chain.

IGMP, ESP, AH, UDPLITE, ALL

Protocol ID: igmp, esp, ah, udplite, all (Table A-11)

***Table A-11.** igmp, esp, ah, udplite, all Variables and Their Expected Values*

Variable Name	Datatype	Semantics
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP addresss
srcipfromo	IP_ADDR	Start of range of source IP address
srcipto	IP_ADDR	End of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
dscp	UINT8 (0x0-0x3f, 0 - 63)	Differentiated Services Code Point
comment (since 0.8.5)	STRING	Text with a maximum of 256 characters

(*continued*)

Table A-11. *(continued)*

Variable Name	Datatype	Semantics
state (since 0.8.5)	STRING	Comma-separated list of NEW, ESTABLISHED, RELATED, INVALID, or NONE
ipset (since 0.9.13)	STRING	The name of an IPSet managed outside of libvirt
ipsetflags (since 0.9.13)	IPSETFLAGS	Flags for the IPSet; requires ipset attribute

Note: The chain parameter is ignored for this type of traffic and should be either omitted or set to the root chain.

TCP/UDP/SCTP over IPV6

Protocol ID: tcp-ipv6, udp-ipv6, sctp-ipv6 (Table A-12)

Table A-12. *tcp-ipv6, udp-ipv6, sctp-ipv6 Variables and Their Expected Values*

Variable Name	Datatype	Semantics
srcmacaddr	MACV6_ADDR	MAC address of sender
srcipaddr	IPV6_ADDR	Source IP address
srcipmask	IPV6_MASK	Mask applied to source IP address
dstipaddr	IPV6_ADDR	Destination IP address
dstipmask	IPV6_MASK	Mask applied to destination IP address
srcipfrom	IPV6_ADDR	Start of range of source IP address
srcipto	IPV6_ADDR	End of range of source IP address

(continued)

Table A-12. *(continued)*

Variable Name	Datatype	Semantics
`dstipfrom`	`IPV6_ADDR`	Start of range of destination IP address
`dstipto`	`IPV6_ADDR`	End of range of destination IP address
`srcportstart`	`UINT16`	Start of range of valid source ports
`srcportend`	`UINT16`	End of range of valid source port
`dstportstart`	`UINT16`	Start of range of valid destination ports
`dstportend`	`UINT16`	End of range of valid destination ports
`dscp`	`UINT8` (0x0-0x3f, 0–63)	Differentiated Services Code Point
`comment` (since 0.8.5)	`STRING`	Text with a maximum of 256 characters
`state` (since 0.8.5)	`STRING`	Comma-separated list of `NEW`, `ESTABLISHED`, `RELATED`, `INVALID`, or `NONE`
`flags` (since 0.9.1)	`STRING`	TCP-only: Format of mask/flags with mask and flags, with each being a comma-separated list of `SYN`, `ACK`, `URG`, `PSH`, `FIN`, `RST`, `NONE`, or `ALL6`
`ipset` (since 0.9.13)	`STRING`	The name of an IPSet managed outside of libvirt
`ipsetflags` (since 0.9.13)	`IPSETFLAGS`	Flags for the IPSet; requires `ipset` attribute

Note: The chain parameter is ignored for this type of traffic and should either be omitted or set to the `root` chain.

ICMPv6

Protocol ID: icmpv6 (Table A-13)

Table A-13. icmpv6 Variables and Their Expected Values

Variable Name	Datatype	Semantics
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacaddr	MACV6_ADDR	MAC address of sourcen
srcmacmask	MACV6_MASK	Mask applied to MAC address of source
dstrcipaddr	IPV6_ADDR	Destination IP destination
dstipmask	IPV6_MASK	Mask applied to IP destination
srcipfrom	IPV6_ADDR	Source IP address
srcipto	IPV6_ADDR	Mask applied to source IP addresss
dstipfrom	IPV6_ADDR	Start of range of destination IP address
dstipto	IPV6_ADDR	End of range of destination IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
type	UINT16	ICMP type
code	UINT16	ICMP code
dscp	UINT8 (0x0-0x3f, 0–63)	Differentiated Services Code Point
comment (since 0.8.5)	STRING	Text with maximum of 256 characters

(*continued*)

Table A-13. *(continued)*

Variable Name	Datatype	Semantics
`state` (since 0.8.5)	`STRING`	Comma-separated list of `NEW`, `ESTABLISHED`, `RELATED`, `INVALID`, or `NONE`
`ipset` (since 0.9.13)	`STRING`	The name of an IPSet managed outside of libvirt
`ipsetflags` (since 0.9.13)	`IPSETFLAGS`	flags for the IPSet; requires `ipset` attribute

Note: The chain parameter is ignored for this type of traffic and should be either omitted or set to the `root` chain.

IGMP, ESP, AH, UDPLITE, ALL over IPv6

Protocol ID: `esp-ipv6`, `ah-ipv6`, `udplite-ipv6`, `all-ipv6` (Table A-14)

Table A-14. *esp-ipv6, ah-ipv6, udplite-ipv6, all-ipv6 Variables and Their Expected Values*

Variable Name	Datatype	Semantics
`srcmacaddr`	`MAC_ADDR`	MAC address of sender
`srcipaddr`	`IPV6_ADDR`	Source IP address
`srcipmask`	`IPV6_MASK`	Mask applied to source IP address
`dstipaddr`	`IPV6_ADDR`	Destination IP address
`dstipmask`	`IPV6_MASK`	Mask applied to destination IP addresss
`srcipfromo`	`IPV6_ADDR`	Start of range of source IP address

(continued)

Table A-14. (*continued*)

Variable Name	Datatype	Semantics
`srcipto`	`IPV6_ADDR`	End of range of source IP address
`dstipfrom`	`IPV6_ADDR`	Start of range of destination IP address
`dstipto`	`IPV6_ADDR`	End of range of destination IP address
`dscp`	`UINT8` (`0x0-0x3f`, `0–63`)	Differentiated Services Code Point
`comment` (since 0.8.5)	`STRING`	Text with maximum of 256 characters
`state` (since 0.8.5)	`STRING`	Comma-separated list of `NEW`, `ESTABLISHED`, `RELATED`, `INVALID`, or `NONE`
`ipset` (since 0.9.13)	`STRING`	The name of an IPSet managed outside of libvirt
`ipsetflags` (since 0.9.13)	`IPSETFLAGS`	Flags for the IPSet; requires `ipset` attribute

Note: The chain parameter is ignored for this type of traffic and should be either omitted or set to the `root` chain.

Advanced Filter Configuration Topics

The following sections discuss advanced filter configuration topics.

Connection Tracking

The network filtering subsystem (on Linux) makes use of the connection tracking support of iptables. This helps enforce the directionality of network traffic (state match) as well as counting and limiting the number of simultaneous connections toward a VM. As an example, if a VM has TCP port 8080 open as a server, clients may connect to the VM on port 8080. Connection tracking and enforcement of directionality then prevents the VM from initiating a connection from (TCP client) port 8080 to the host back to a remote host. More importantly, tracking helps to prevent remote attackers from establishing a connection back to a VM. For example, if the user inside the VM established a connection to port 80 on an attacker site, then the attacker will not be able to initiate a connection from TCP port 80 back toward the VM. By default, the connection state match enables connection tracking, and then enforcement of directionality of traffic is turned on.

Listing A-76 shows an example XML fragment where this feature has been turned off for incoming connections to TCP port 12345.

Listing A-76. Turning Off Port 12345

```
...
<rule direction='in' action='accept' statematch='false'>
  <tcp dstportstart='12345'/>
</rule>
...
```

This now allows incoming traffic to TCP port 12345 but also enables the initiation from (client) TCP port 12345 within the VM, which may or may not be desirable.

Limiting Number of Connections

To limit the number of connections a VM may establish, a rule must be provided that sets a limit of connections for a given type of traffic. If, for example, a VM is supposed to be allowed to ping only one other IP address at a time and is supposed to have only one active incoming SSH connection at a time, the XML fragment in Listing A-77 can be used to achieve this.

Listing A-77. Limiting Connections to a Port

```
...
<rule action='drop' direction='in' priority='400'>
  <tcp connlimit-above='1'/>
</rule>
<rule action='accept' direction='in' priority='500'>
  <tcp dstportstart='22'/>
</rule>
<rule action='drop' direction='out' priority='400'>
  <icmp connlimit-above='1'/>
</rule>
<rule action='accept' direction='out' priority='500'>
  <icmp/>
</rule>
<rule action='accept' direction='out' priority='500'>
  <udp dstportstart='53'/>
</rule>
<rule action='drop' direction='inout' priority='1000'>
  <all/>
</rule>
...
```

Note that the rule for the limit has to logically appear before the rule for accepting the traffic.

An additional rule for letting DNS traffic to port 22 go out the VM has been added to avoid SSH sessions not getting established for reasons related to DNS lookup failures by the SSH daemon. Leaving this rule out may otherwise lead to fun-filled debugging joy (because the SSH client seems to hang while trying to connect).

A lot of care must be taken with timeouts related to tracking traffic. An ICMP ping that the user may have terminated inside the VM may have a long timeout in the host's connection tracking system and therefore not allow another ICMP ping to go through for a while. Therefore, the timeouts have to be tuned in the host's sysfs (Listing A-78).

Listing A-78. Finding the Host's Timeouts

```
echo 3 > /proc/sys/net/netfilter/nf_conntrack_icmp_timeout
```

This sets the ICMP connection tracking timeout to three seconds. The effect of this is that once one ping is terminated, another one can start after three seconds. Further, I want to point out that a client that for whatever reason has not properly closed a TCP connection may cause a connection to be held open for a longer period of time, depending on what timeout the `TCP established` state timeout has been set to on the host. Also, idle connections may time out in the connection tracking system but can be reactivated once packets are exchanged. However, a newly initiated connection may force an idle connection into TCP. If the number of allowed connections is set to a too-low limit, the new connection is established and hits (not exceeds) the limit of allowed connections. For example, a key is pressed on the old SSH session, which now has become unresponsive because of its traffic being dropped. Therefore, the limit of connections should be rather high so that fluctuations in new TCP connections don't cause odd traffic behavior in relation to idle connections.

Command-Line Tools

Table A-15 lists the example network filters that are automatically installed with libvirt.

Table A-15. *Command-Line Tools*

Name	Description
`no-arp-spoofing`	Prevent a VM from spoofing ARP traffic; this filter only allows ARP request and reply messages and enforces that those packets contain the MAC and IP addresses of the VM.
`allow-dhcpg`	Allow a VM to request an IP address via DHCP (from any DHCP server).
`allow-dhcp-server`	Allow a VM to request an IP address from a specified DHCP server. The dotted decimal IP address of the DHCP server must be provided in a reference to this filter. The name of the variable must be `DHCPSERVER`.
`no-ip-spoofing`	Prevent a VM from sending IP packets with a source IP address different from the one in the packet.
`no-ip-multicast`	Prevent a VM from sending IP multicast packets.
`clean-traffic`	Prevent MAC, IP, and ARP spoofing. This filter references several other filters as building blocks.

Note that most of these filters are only building blocks and require a combination with other filters to provide useful network traffic filtering. The most useful one in Table A-15 is the `clean-traffic` filter. This filter itself can, for example, be combined with the `no-ip-multicast` filter to prevent virtual machines from sending IP multicast traffic on top of the prevention of packet spoofing.

Writing Your Own Filters

Since libvirt provides only a couple of example networking filters, you may consider writing your own. When doing so, there are a couple of things you need to know regarding the network filtering subsystem and how it works internally. Certainly you also have to know and understand the protocols well that you want to be filtering on so that no further traffic than what you want can pass and that in fact the traffic you want to allow does pass.

The network filtering subsystem is currently available only on Linux hosts and works only for the QEMU and KVM types of virtual machines. On Linux it builds upon the support for ebtables, iptables, and ip6tables and makes use of their features. From the previous list of supported protocols, the following ones are implemented using ebtables:

- `mac`
- `stp`
- `vlan` (802.1Q)
- `arp`, `rarp`
- `ipv4`
- `ipv6mac`

All other protocols over IPv4 are supported using iptables; those over IPv6 are implemented using ip6tables.

On a Linux host, all traffic filtering instantiated by libvirt's network filter subsystem first passes through the filtering support implemented by ebtables and only then through the iptables or ip6table filters. If a filter tree has rules with the protocols `mac`, `stp`, `vlan` `arp`, `rarp`, `ipv4`, or `ipv6`, ebtables rules will automatically be instantiated.

The role of the `chain` attribute in the network filter XML is that internally a new user-defined ebtables table is created that then for example receives all ARP traffic coming from or going to a virtual machine if the chain `arp` has been specified. Further, a rule is generated in an interface's root chain that directs all IPv4 traffic into the user-defined chain. Therefore, all ARP traffic rules should then be placed into filters specifying this chain. This type of branching into user-defined tables is only supported with filtering on the ebtables layer.

Since 0.9.8, multiple chains for the same protocol can be created. For this, the name of the chain must have a prefix of one of the previously enumerated protocols. To create an additional chain for handling ARP traffic, a chain with name `arp-test` can be specified.

As an example, it is possible to filter on UDP traffic by source and destination ports using the `ip` protocol filter and specifying attributes for the protocol, source, and destination IP addresses and ports of UDP packets that are to be accepted. This allows early filtering of UDP traffic with ebtables. However, once an IP or IPv6 packet, such as a UDP packet, has passed the ebtables layer and there is at least one rule in a filter tree that instantiates the iptables or ip6tables rules, a rule to let the UDP packet pass will also be necessary to be provided for those filtering layers. This can be achieved with a rule containing an appropriate `udp` or `udp-ipv6` traffic filtering node.

Custom Filter 1

As an example, say you want to now build a filter that fulfills the following list of requirements:

- Prevents a VM's interface from MAC, IP, and ARP spoofing
- Prevents a VM's interface from MAC, IP, and ARP spoofing
- Allows the VM to send ping traffic from an interface but not let the VM be pinged on the interface
- Allows the VM to do DNS lookups (UDP toward port 53)

The requirement to prevent spoofing is fulfilled by the existing `clean-traffic` network filter; thus, you will reference this filter from your custom filter.

To enable traffic for TCP ports 22 and 80, you will add two rules to enable this type of traffic. To allow the VM to send ping traffic, you will add a rule for ICMP traffic. For simplicity reasons, you allow general ICMP traffic to be initiated from the VM, not just ICMP echo request and response messages. To then disallow all other traffic to reach or be initiated by the VM, you will then need to add a rule that drops all other traffic. Assuming the VM is called `test` and the interface you want to associate your filter with is called `eth0`, you can name your filter `test-eth0`. The result of these considerations is shown in Listing A-79.

Listing A-79. Enabling TCP Ports 22 and 80

```
<filter name='test-eth0'>
  <!-- reference the clean traffic filter to prevent
       MAC, IP and ARP spoofing. By not providing
       and IP address parameter, libvirt will detect the
       IP address the VM is using. -->
  <filterref filter='clean-traffic'/>

  <!-- enable TCP ports 22 (ssh) and 80 (http) to be reachable
-->
  <rule action='accept' direction='in'>
    <tcp dstportstart='22'/>
  </rule>

  <rule action='accept' direction='in'>
    <tcp dstportstart='80'/>
  </rule>

  <!-- enable general ICMP traffic to be initiated by the VM;
       this includes ping traffic -->
  <rule action='accept' direction='out'>
    <icmp/>
  </rule>

  <!-- enable outgoing DNS lookups using UDP -->
  <rule action='accept' direction='out'>
    <udp dstportstart='53'/>
  </rule>

  <!-- drop all other traffic -->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>

</filter>
```

Note that none of the rules in the previous XML contains the IP address of the VM as either the source or destination address, yet the filtering of the traffic works correctly. The reason is that the evaluation of the rules internally happens on a per-interface basis, and the rules are evaluated based on the knowledge about which (tap) interface has sent or will receive the packet rather than what their source or destination IP address may be.

An XML fragment for a possible network interface description inside the domain XML of the test VM could then look like Listing A-80.

Listing A-80. Specifying an Interface Description

```
...
<interface type='bridge'>
  <source bridge='mybridge'/>
  <filterref filter='test-eth0'/>
</interface>
...
```

To more strictly control the ICMP traffic and enforce that only ICMP echo requests can be sent from the VM and only ICMP echo responses be received by the VM, the previous ICMP rule can be replaced with the two rules in Listing A-81.

Listing A-81. Enabling Outgoing ICMP Requests

```
<!-- enable outgoing ICMP echo requests-->
<rule action='accept' direction='out'>
  <icmp type='8'/>
</rule>

<!-- enable incoming ICMP echo replies-->
<rule action='accept' direction='in'>
  <icmp type='0'/>
</rule>
```

Custom Filter 2

In this example, you now want to build a similar filter as in the previous example but extend the list of requirements with an FTP server located inside the VM. Further, you will be using features that have been added in version 0.8.5. The requirements for this filter are as follows:

- Prevents a VM's interface from MAC, IP, and ARP spoofing
- Opens only TCP ports 22 and 80 of a VM's interface
- Allows the VM to send ping traffic from an interface but not let the VM be pinged on the interface
- Allows the VM to do DNS lookups (UDP toward port 53)
- Enable an FTP server (in active mode) to be run inside the VM

The additional requirement of allowing an FTP server to be run inside the VM maps into the requirement of allowing port 21 to be reachable for FTP control traffic as well as enabling the VM to establish an outgoing TCP connection originating from the VM's TCP port 20 back to the FTP client (FTP active mode). There are several ways this filter can be written; two solutions are presented here.

The first solution makes use of the `state` attribute of the TCP protocol that gives you a hook into the connection tracking framework of the Linux host. For the VM-initiated FTP data connection (FTP active mode), you use the `RELATED` state that allows you to detect that the VM-initiated FTP data connection is a consequence of (or "has a relationship with") an existing connection you control; thus, you want to allow it to let packets pass the firewall. The `RELATED` state, however, is valid only for the first packet of the outgoing TCP connection for the FTP data path. Afterward, the state to compare against is `ESTABLISHED`, which then applies equally to the incoming and outgoing direction. All this is related to the FTP data traffic originating from TCP port 20 of the VM. This then leads to the solution (since version 0.8.5; QEMU, KVM) shown in Listing A-82.

Listing A-82. Network Filter for FTP Traffic

```
<filter name='test-eth0'>
  <!-- reference the clean traffic filter to prevent
       MAC, IP and ARP spoofing. By not providing
       and IP address parameter, libvirt will detect the
       IP address the VM is using. -->
  <filterref filter='clean-traffic'/>

  <!-- enable TCP port 21 (ftp-control) to be reachable -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='21'/>
  </rule>
  <!-- enable TCP port 20 for VM-initiated ftp data connection
       related to an existing ftp control connection -->
  <rule action='accept' direction='out'>
    <tcp srcportstart='20' state='RELATED,ESTABLISHED'/>
  </rule>

  <!-- accept all packets from client on the ftp data
  connection -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='20' state='ESTABLISHED'/>
  </rule>

  <!-- enable TCP ports 22 (ssh) and 80 (http) to be reachable -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='22'/>
  </rule>

  <rule action='accept' direction='in'>
    <tcp dstportstart='80'/>
  </rule>

  <!-- enable general ICMP traffic to be initiated by the VM;
       this includes ping traffic -->
  <rule action='accept' direction='out'>
    <icmp/>
  </rule>

  <!-- enable outgoing DNS lookups using UDP -->
  <rule action='accept' direction='out'>
    <udp dstportstart='53'/>
  </rule>
  <!-- drop all other traffic -->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>

</filter>
```

Before trying a filter using the RELATED state, you have to make sure that the appropriate connection tracking module has been loaded into the host's kernel. Depending on the version of the kernel, you must run either one of the two commands in Listing A-83 before the FTP connection with the VM is established.

Listing A-83. modprob Commands

```
modprobe nf_conntrack_ftp   # where available  or
modprobe ip_conntrack_ftp   # if above is not available
```

If other protocols than FTP are to be used in conjunction with the RELATED state, their corresponding module must be loaded. Modules exist at least for the protocols ftp, tftp, irc, sip, sctp, and amanda.

The second solution makes uses the state flags of connections more than the previous solution did. In this solution, you take advantage of the fact that the NEW state of a connection is valid when the first packet of a traffic flow is seen. Subsequently, if the first packet of a flow is accepted, the flow becomes a connection and enters the ESTABLISHED state. This allows you to write a general rule for allowing packets of ESTABLISHED connections to reach the VM or be sent by the VM. You write specific rules for the first packets identified by the NEW state and for which ports they are acceptable. All packets for ports that are not explicitly accepted will be dropped, and therefore the connection will not go into the ESTABLISHEDD state, and any subsequent packets be dropped (Listing A-84) .

Listing A-84. A clean-traffic Filter to Prevent Spoofing

```
<filter name='test-eth0'>
  <!-- reference the clean traffic filter to prevent
       MAC, IP and ARP spoofing. By not providing
       and IP address parameter, libvirt will detect the
       IP address the VM is using. -->
  <filterref filter='clean-traffic'/>

  <!-- let the packets of all previously accepted connections
  reach the VM -->
  <rule action='accept' direction='in'>
    <all state='ESTABLISHED'/>
  </rule>

  <!-- let the packets of all previously accepted and related
  connections be sent from the VM -->
  <rule action='accept' direction='out'>
    <all state='ESTABLISHED,RELATED'/>
  </rule>

  <!-- enable traffic towards port 21 (ftp), 22 (ssh) and 80
  (http) -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='21' dstportend='22' state='NEW'/>
  </rule>

  <rule action='accept' direction='in'>
    <tcp dstportstart='80' state='NEW'/>
  </rule>

  <!-- enable general ICMP traffic to be initiated by the VM;
       this includes ping traffic -->
  <rule action='accept' direction='out'>
```

```
    <icmp state='NEW'/>
  </rule>

  <!-- enable outgoing DNS lookups using UDP -->
  <rule action='accept' direction='out'>
    <udp dstportstart='53' state='NEW'/>
  </rule>

  <!-- drop all other traffic -->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>

</filter>
```

Limitations

The following sections list the (current) limitations of the network filtering subsystem.

VM Migration

VM migration is supported only if the whole filter tree that is referenced by a virtual machine's top-level filter is also available on the target host. The network filter `clean-traffic`, for example, should be available on all libvirt installations of version 0.8.1 or later and thus enable migration of VMs that, for example, reference this filter. All other custom filters must be migrated using higher-layer software. It is outside the scope of libvirt to ensure that referenced filters on the source system are equivalent to those on the target system and vice versa.

Migration must occur between libvirt installations of version 0.8.1 or newer not to lose the network traffic filters associated with an interface.

VLAN Filtering on Linux

VLAN (802.1Q) packets, if sent by a virtual machine, cannot be filtered with rules for the protocol IDs `arp`, `rarp`, `ipv4`, and `ipv6` but only with the protocol IDs `mac` and `vlan`. Therefore, the example filter `clean-traffic` will not work as expected.

Index