Running OpenWRT ARM under QEMU

openwrt-qemu.md

Environment

The steps shown below are done on a Ubuntu VM using Qemu 3.0

```
$ qemu-system-arm -version
QEMU emulator version 3.0.0
Copyright (c) 2003-2017 Fabrice Bellard and the QEMU Project developers
```

To quit Qemu at any time press `Ctrl+a x`, i.e. `Ctrl+a` and then `x`

Using a prebuilt image

The latest OpenWRT version at the time of writing is 18.06.1 available at https://downloads.openwrt.org/releases/18.06.1/. The armvirt target is the one intented for emulation and we'll use that.

1. Using initramfs

Here the rootfs is bundled along with the zImage as a single file. In this mode the filesystem resides entirely in memory and any modifications are lost on poweroff.

Download the bundled zImage

```
$ wget -q
https://downloads.openwrt.org/releases/18.06.1/targets/armvirt/32/openwrt-
```

```
18.06.1-armvirt-32-zImage-initramfs -O zImage-initramfs
```

and run

```
$ qemu-system-arm -M virt -kernel zImage-initramfs -no-reboot -
nographic
```

2. Using a separate rootfs

Here the rootfs is present as a separate cpio archive.

Download plain zImage and rootfs as separate files

```
$ wget -q
https://downloads.openwrt.org/releases/18.06.1/targets/armvirt/32/openwrt-
18.06.1-armvirt-32-zImage -O zImage
$ wget -q
https://downloads.openwrt.org/releases/18.06.1/targets/armvirt/32/openwrt-
18.06.1-armvirt-32-rootfs.cpio.gz -O rootfs.cpio.gz
$ gunzip rootfs.cpio.gz
```

and run

```
$ qemu-system-arm -M virt -kernel zImage -initrd rootfs.cpio -no-reboot
-nographic
```

Setting up networking

The Qemu VM setup doesn't have network access at the moment, i.e. both outgoing and incoming connections will not work. First let's setup outgoing connections i.e from the OpenWRT VM to the outside world.

Download the squashfs filesystem

```
$ wget -q
https://downloads.openwrt.org/releases/18.06.1/targets/armvirt/32/openwrt-
18.06.1-armvirt-32-root.squashfs.gz -O root.squashfs.gz
$ gunzip root.squashfs.gz
```

The squashfs filesystem contains the necessary kernel modules for setting up proper networking. OpenWRT requires **atleast two** NIC's and correspondingly need to provide appropriate command line args to Qemu.

- `eth0` acts as a LAN port
- `eth1` acts as a WAN port providing Internet access.

We can have multiple LAN ports if necessary. This is equivalent to Ethernet LAN ports in routers (4 ports in most of them) where we connect our equipment. In OpenWRT all LAN ports are connected together using a bridge - the `br-lan` interface.

Run Qemu

```
$ qemu-system-arm -M virt-2.9 \
 -kernel zImage \
 -no-reboot -nographic \
 -nic user -nic user \
 -drive file=root.squashfs,if=virtio,format=raw \
 -append "root=/dev/vda"
```

- The `initrd` parameter is not required anymore and may be omitted
- The machine (`-M` parameter) must be one of `virt-2.6`, `virt-2.7`, `virt-2.8` or `virt-2.9`. In my tests, using `virt` or `virt-3.0` doesn't work.
- The `-nic` parameters adds the two network cards

When Qemu is up and running wait until the NIC's are initialized. There will be kernel messages similar to the one below.

```
[   30.083201] br-lan: port 1(eth0) entered blocking state
[   30.083742] br-lan: port 1(eth0) entered disabled state
[   30.087443] device eth0 entered promiscuous mode
[   30.185675] br-lan: port 1(eth0) entered blocking state
[   30.186191] br-lan: port 1(eth0) entered forwarding state
[   30.187786] IPv6: ADDRCONF(NETDEV_UP): br-lan: link is not ready
[   30.907054] IPv6: ADDRCONF(NETDEV_CHANGE): br-lan: link becomes
ready
[   31.166951] 8021q: adding VLAN 0 to HW filter on device eth1
```

Execute `ifconfig`. There should be two network interfaces `eth0` and `eth1` with an IP assigned on the latter like the snippet shown below.

```
root@OpenWrt:/# ifconfig
br-lan    Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:192.168.1.1  Bcast:192.168.1.255
```

```
Mask:255.255.255.0
          inet6 addr: fd25:fc2:e636::1/60 Scope:Global
          inet6 addr: fe80::5054:ff:fe12:3456/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:22 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1152 (1.1 KiB)  TX bytes:3188 (3.1 KiB)

eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:29 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1290 (1.2 KiB)  TX bytes:4491 (4.3 KiB)

eth1      Link encap:Ethernet  HWaddr 52:54:00:12:34:57
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fec0::5054:ff:fe12:3457/64 Scope:Site
          inet6 addr: fe80::5054:ff:fe12:3457/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1290 (1.2 KiB)  TX bytes:2152 (2.1 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Further to check if the VM can reach the internet do a `wget` (`curl` isn't available) like below

```
root@OpenWrt:/# wget -q http://www.icanhazip.com -O -
35.229.164.189
```

At this point connections from the VM to the outside world work. To allow incoming connections we can use the port forwarding feature of Qemu.

Let's say we want to ssh connections to the VM. SSH runs on Port 22. We want to forward host port 2222 to guest port 22.

```
$ qemu-system-arm -M virt-2.9 \
 -kernel zImage \
 -no-reboot -nographic \
 -device virtio-net-pci \
 -netdev user,id=net1,hostfwd=tcp::2222-:22 -device virtio-net-
pci,netdev=net1 \
 -drive file=root.squashfs,if=virtio,format=raw \
 -append "root=/dev/vda"
```

Here we have to specify both the frontend (qemu side) and backend (host side) of the network using the `-device` and `-netdev` parameters respectively. Note that although we have added two NICs (`-device`) corresponding to `eth0` and `eth1` only the latter is connected to the Qemu network backend. This is equivalent to a router which has no PCs connected to its LAN ports with the WAN side connected to the internet.

The `-nic` parameter which we used before was a simple way to set up both sides using a single command. ~~However it doesn't support advanced cases like this when we need to port forward.~~ The `-nic` parameter does support specifying advanced parameters (example `-nic user,hostfwd=tcp::2222-:22`).

There's is still one more thing left before we can ssh to our VM. OpenWRT comes with a firewall which by default doesn't allow any incoming connections. Adding a firewall rule to allow SSH on port 22 is the best thing to do but for simplicity let's disable the firewall altogether.

```
root@OpenWrt:/# /etc/init.d/firewall stop
Warning: Unable to locate ipset utility, disabling ipset support
 * Flushing IPv4 filter table
 * Flushing IPv4 nat table
 * Flushing IPv4 mangle table
 * Flushing IPv6 filter table
 * Flushing IPv6 mangle table
 * Flushing conntrack table ...
```

Now we can SSH on port 2222 which is forwarded to Port 22 in Qemu.

```
ec@vm $ ssh -p2222 root@127.0.0.1
The authenticity of host '[127.0.0.1]:2222 ([127.0.0.1]:2222)' can't be
established.
RSA key fingerprint is e8:38:55:a6:04:28:0e:f2:f0:38:fd:30:c1:ab:4c:df.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[127.0.0.1]:2222' (RSA) to the list of
known hosts.
```

```
BusyBox v1.28.3 () built-in shell (ash)


  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 OpenWrt 18.06.1, r7258-5eb055306f
 -----------------------------------------------------
root@OpenWrt:~#
```

Customizing OpenWRT

The precompiled images are helpful to quickly set up an OpenWRT VM but they lack some important packages like `curl` or even `python` if we want to run some scripts. To customize OpenWRT to our liking there are two ways:

- Use the built-in package manager `opkg` to install the necessary packages
- Compile OpenWRT and generate an image containing all the packages we need

Installing packages on-board

If we follow the first method we cannot use the `squashfs` file as the hard disk as it's a read-only filesystem. Rather we need to use the `ext4` filesystem provided at the same page.

```
$ wget -q
https://downloads.openwrt.org/releases/18.06.1/targets/armvirt/32/openwrt-
18.06.1-armvirt-32-root.ext4.gz -O root.ext4.gz
$ gunzip root.ext4.gz
$ qemu-system-arm -M virt-2.9  -kernel zImage  -no-reboot -nographic  -
device virtio-net-pci  -netdev user,id=net1,hostfwd=tcp::2222-:22 -
device virtio-net-pci,netdev=net1  -drive
file=root.ext4,if=virtio,format=raw  -append "root=/dev/vda"
```

We can now install our favourite packages like `curl` and `python`.

```
root@OpenWrt:/# opkg update
root@OpenWrt:/# opkg install curl python
```

Compiling OpenWRT to include the necessary packages

OpenWRT ships with ImageBuilder (openwrt-imagebuilder-armvirt.tar.xz) a tool to generate images as per our liking. We can easily customize the generated image by including new packages or by adding/changing config files (like disabling firewall at boot). Using the imagebuilder is a fast alternative as opposed to using the SDK(openwrt-sdk-armvirt.tar.xz) which requires a full compiling from source of everything including the packages.

Further, since the packages are amalgamated within the base image we can use a read-only filesystem like `squashfs`. The steps below show how we can compile OpenWRT including the packages curl and python.

```
$ wget -q
https://downloads.openwrt.org/releases/18.06.1/targets/armvirt/32/openwrt-
imagebuilder-18.06.1-armvirt-32.Linux-x86_64.tar.xz
$ tar -jxf openwrt-imagebuilder-18.06.1-armvirt-32.Linux-x86_64.tar.xz
$ cd openwrt-imagebuilder-18.06.1-armvirt-32.Linux-x86_64.tar/
$ make image PACKAGES="curl python"
```

To remove a package, for example to save space, prepend the package name with a hyphen.

```
$ make image PACKAGES="-dropbear"
```

If we want to include any files in the generated image we can do like

```
$ mkdir -p files/tmp
$ touch files/tmp/myfile
$ make image FILES=files/
```

`myfile` will be located under `/tmp` in the generated image. If this conflicts with an existing file our new file will take preference. Thus using this technique we can override the default config files that are shipped with OpenWRT. The generated images are located in the bin directory and can be booted in Qemu as we did before.

References

- https://openwrt.org/docs/guide-user/additional-software/imagebuilder

- https://openwrt.org/docs/guide-developer/quickstart-build-images
- https://openwrt.org/docs/guide-user/additional-software/beginners-build-guide
- https://openwrt.org/docs/guide-user/virtualization/qemu
- https://wiki.openwrt.org/doc/howto/qemu
- https://blog.ljdelight.com/compiling-openwrt-from-source/
- https://wiki.openwrt.org/doc/howto/obtain.firmware.generate
- https://wiki.openwrt.org/doc/howto/obtain.firmware
- https://wiki.openwrt.org/doc/techref/filesystems
- https://openwrt.org/packages/start
- https://downloads.lede-project.org/releases/17.01.4/targets/armvirt/generic/
- https://github.com/lede-project/source/tree/master/target/linux/armvirt
- https://wiki.openwrt.org/doc/networking/network.interfaces
- https://wiki.openwrt.org/doc/faq/after.installation
- https://wiki.openwrt.org/doc/uci/firewall
- https://openwrt.org/docs/guide-user/firewall/firewall_configuration
- https://www.qemu.org/2018/05/31/nic-parameter/

extremecoders-re commented on Dec 20, 2018

Notes for MIPS

Issue

For MIPS, the ImageBuilder does not generate the root fs. Although the download page does provide a pre-compiled kernel with embedded initramfs (vmlinux-initramfs.elf), however the ImageBuilder does not seem to generate a root fs either embedded in the kernel as initramfs or separate. Without the initramfs, the kernel is not of much use.

Solution

The workaround is to generate the root fs manually. Steps are listed below.

1. Use ImageBuilder as you would normally

```
$ pwd
/home/ubuntu/workspace/malta/openwrt-imagebuilder-18.06.1-malta-be.Linux-x86_64
$ make image PACKAGES="python"
```

2. Use `mksquashfs` to generate the root fs manually. Use `xz` compression.

```
$ pwd
/home/ubuntu/workspace/malta/openwrt-imagebuilder-18.06.1-malta-be.Linux-
x86_64/bin/targets/malta/be
$ mksquashfs ../../../../build_dir/target-mips_24kc_musl/root-malta/
rootfs.squashfs -comp xz
```

3. And boot!

```
$ ls
openwrt-18.06.1-malta-be-default.manifest
openwrt-18.06.1-malta-be-uImage-gzip
openwrt-18.06.1-malta-be-uImage-lzma
openwrt-18.06.1-malta-be-vmlinux.elf*
rootfs.squashfs
sha256sums

$ qemu-system-mips -M malta \
-kernel openwrt-18.06.1-malta-be-vmlinux.elf \
-drive file=rootfs.squashfs,format=raw \
-append "root=/dev/sda" \
-nographic -no-reboot
```

extremecoders-re commented on Nov 13, 2019

Changing the IP address assigned via DHCP in user mode networking

```
$ qemu-system-arm -M virt-2.9 \
        -kernel zImage \
        -nographic \
        -no-reboot \
        -drive file=rootfs.squashfs,if=virtio,format=raw \
        -append "root=/dev/vda" \
        -nic user \
        -nic user,net=192.168.76.0/24
```

From: https://wiki.qemu.org/Documentation/Networking

extremecoders-re commented on Nov 18, 2019

In general (atleast for ARM), for successful emulation ensure that the Linux kernel version is more recent than the Qemu version.

Example:
The Firmadyne ARM kernel is at version 4.1.17+ which was released on Jan 2016.

The kernel is successfully emulated by Qemu 2.0.0 which was released on April 2014.

However, Qemu 2.11.1 released on Feb 2018 fails to emulate. It gets stuck without printing anything.

The most probable reason is due to the virtio driver. A newer Linux kernel will support an older virto driver in Qemu and not the other way round.