Understanding TUN TAP Interfaces

Posted on 17th October 2014 by doozer

As I mentioned in the previous article Understanding Bridges, Linux and most other operating systems have the ability to create virtual interfaces which are usually called TUN/TAP devices. This article will discuss those devices with particular focus on how they are used in OpenStack.

Typically a network device in a system, for example eth0, has a physical device associated with it which is used to put packets on the wire. In contrast a TUN or a TAP device is entirely virtual and managed by the kernel. User space applications can interact with TUN and TAP devices as if they were real and behind the scenes the operating system will push or inject the packets into the regular networking stack as required making everything appear as if a real device is being used.

You might wonder why there are two options, surely a network device is a network device and that's the end of the story. That's partially true but TUN and TAP devices aim to solve different problems.

TUN Interfaces

TUN devices work at the IP level or layer three level of the network stack and are usually point-to-point connections. A typical use for a TUN device is establishing VPN connections since it gives the VPN software a chance to encrypt the data before it gets put on the wire. Since a TUN device works at layer three it can only accept IP packets and in some cases only IPv4. If you need to run any other protocol over a TUN device you're out of luck. Additionally because TUN devices work at layer three they can't be used in bridges and don't typically support broadcasting.

TAP Interfaces

TAP devices, in contrast, work at the Ethernet level or layer two and therefore behave very much like a real network adaptor. Since they are running at layer two they can transport any layer three protocol and aren't limited to point-to-point connections. TAP devices can be part of a bridge and are commonly used in virtualization systems to provide virtual network adaptors to multiple guest machines. Since TAP devices work at layer two they will forward broadcast traffic which normally makes them a poor choice for VPN connections as the VPN link is typically much narrower than a LAN network (and usually more expensive).

Managing Virtual Interfaces

It really couldn't be simpler to create a virtual interface:

```
ip tuntap add name tap0 mode tap
ip link show
```

The above command creates a new TAP interface called tap0 and then shows some information about the device. You will probably notice that after creation the tap0 device reports that it is in the down state. This is by design and it will come up only when something binds it (see here). The output of the show command will look something like this:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group
 link/ether 08:00:27:4a:5e:e1 brd ff:ff:ff:ff:ff:ff
3: tap0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 50
 link/ether 36:2b:9d:5c:92:78 brd ff:ff:ff:ff:ff:ff
```

To remove a TUN/TAP interface just replace "add" in the creation command with "del". Note that you have to specify the mode when deleting,

presumably you can create both a tun and a tap interface with the same name.

Creating Veth Pairs

A pair of connected interfaces, commonly known as a veth pair, can be created to act as virtual wiring. Essentially what you are creating is a virtual equivalent of a patch cable. What goes in one end comes out the other. The command to create a veth pair is a little more complicated than some:

```
ip link add ep1 type veth peer name ep2
```

This will create a pair of linked interfaces called *ep1* and *ep2* (ep for Ethernet pair, you probably want to choose more descriptive names). When working with OpenStack, especially on a single box install, it's common to use veth pairs to link together the internal bridges. It is also possible to add IP addresses to the interfaces, for example:

```
ip addr add 10.0.0.10 dev ep1
ip addr add 10.0.0.11 dev ep2
```

Now you can use *ip address show* to check the assignment of IP addresses which will output something like this:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
 inet 127.0.0.1/8 scope host lo
 valid_lft forever preferred_lft forever
 inet6 ::1/128 scope host
 valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
 link/ether 08:00:27:4a:5e:e1 brd ff:ff:ff:ff:ff:ff
 inet 192.168.1.141/24 brd 192.168.1.255 scope global eth0
 valid_lft forever preferred_lft forever
 inet6 fe80::a00:27ff:fe4a:5ee1/64 scope link
 valid_lft forever preferred_lft forever
4: ep2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
 link/ether fa:d3:ce:c3:da:ad brd ff:ff:ff:ff:ff:ff
 inet 10.0.0.11/32 scope global ep2
 valid_lft forever preferred_lft forever
5: ep1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
 link/ether e6:80:a3:19:2c:10 brd ff:ff:ff:ff:ff:ff
 inet 10.0.0.10/32 scope global ep1
 valid_lft forever preferred_lft forever
```

Using a couple of parameters on the ping command shows us the veth pair working:

```
ping -I 10.0.0.10 -c1 10.0.0.11
PING 10.0.0.11 (10.0.0.11) from 10.0.0.10 : 56(84) bytes of data.
64 bytes from 10.0.0.11: icmp_seq=1 ttl=64 time=0.036 ms
--- 10.0.0.11 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.036/0.036/0.036/0.000 ms
```

The *-I* parameter specifies the interface that should be used for the ping. In this case the 10.0.0.10 interface what chosen which is a pair with 10.0.0.11 and as you can see the ping is there and back in a flash. Attempting to ping anything external fails since the veth pair is essentially just a patch cable (although ping'ing eth0 works for some reason).

Related