Networking UML using bridging

David Cannings

Networking UML using bridging

David Cannings

This HOWTO covers connecting User-mode Linux sessions to a physical network by means of the 802.1d Ethernet bridging support available in the Linux kernel. Networking UML sessions in this way enables them to appear and behave as if they were a part of the real LAN. A familiarization with Linux and kernel recompilation is assumed.

Table of Contents

Chapter 1. Introduction

When it comes to networking together UML instances there are plenty of options. The switch daemon or multicast allow us to have a purely virtual network and pcap allows us to monitor host traffic, but what if we want our UML sessions to appear as part of an external network? This could be the case if we were trying to provide some form of UML hosting solution for end users or we have service daemons that run inside UML sessions that we want our LAN, or even the internet, to see.

By implementing connectivity using bridging our UML sessions can have public or private (RFC1918) IP addresses and we have no need for any sort of special routing, either on the physical network or inside the UML instance itself. As far as the end-user or any daemons inside the UML are concerned, there is a normal ethernet adapter device the same as on a physical machine.

Chapter 2. The host kernel

We need a host kernel with two extra options in order to configure our setup in the way described in this document.

For those using **`make menuconfig`** the options required are "802.1d Ethernet Bridging" (in "Networking options") and "Universal TUN/TAP device driver support" (in "Network device support").

For those using **`make config`** the two options are `CONFIG_TUN` and `CONFIG_BRIDGE`.

Chapter 3. Helper utilities

In order to manipulate bridges and tun devices we need two programs, "brctl" and "tunctl". There are Debian packages for these which can be installed as follows:

```
# apt-get install bridge-utils uml-utilities
```

Gentoo users can do the following:

```
# emerge bridge-utils usermode-utilities
```

If you don't happen to be using Debian or Gentoo, alternative links are below.

Package	**URL**
Bridge Utils (RPM)	bridge-utils-0.9.6-1.i386.rpm [http://bridge.sourceforge.net/bridge-utils/bridge-utils-0.9.6-1.i386.rpm]
Bridge Utils (Source)	bridge-utils-0.9.6.tar.gz [http://bridge.sourceforge.net/bridge-utils/bridge-utils-0.9.6.tar.gz]
Tun Utils (Source)	UML download page [http://user-mode-linux.sourceforge.net/dl-sf.html]

The links above were correct at the time of writing but it is advisable to check the bridge page [http://bridge.sourceforge.net/download.html] at Sourceforge for the latest tools if you are not using a package manager. If you are using a RPM based distribution it may also be advisable to check and see if the bridge-utils package comes with it. The bridge-utils RPM supplied with Fedora Core 1 is reported to work.

Instructions for compiling the bridging utilities from a source archive can be found in the Bridge-STP-Howto. [http://www.tldp.org/HOWTO/BRIDGE-STP-HOWTO/preparing-the-bridge.html#COMPILE-THE-UTILS]

Chapter 4. Bridging: The theory

A bridge is a device that forwards traffic between two network segments. In very loose terms it could be considered to be similar to a normal hardware switch. Bridges are "intelligent", meaning they learn which addresses are connected to which port on themselves and therefore do not simply repeat traffic to all connected devices. The bridge is transparent however, which means it wont appear in traceroute outputs.

When we initially create a bridge in Linux it isn't connected to anything. By adding devices we effectively plug them into the ports of this bridge and they become a part of it.

Note

An important distinction to make is that the bridge makes decisions based on MAC address and not IP address, as it is a layer 2 device. The bridging code inside the 2.4 kernel appears before the iptables hooks, therefore bridging happens before packets go through filtering. This means that we can't use all of the features of iptables with our bridge. One solution for this is "ebtables", more information and a good diagram of how packets traverse the kernel is available on the website. For 2.6 kernels, this has changed and filtering should be possible using iptables.

Chapter 5. Bridging: In practice

In the example given here there is a physical network of computers using the addresses 192.168.0.X. There is a host machine running UML sessions that has the IP address 192.168.0.100.

Warning

It's quite hard to experiment over SSH to a remote host unless you have two ethernet cards and can SSH through one whilst configuring the other. In order to attach our ethernet device (ethX) to the bridge we need to assign it an IP of `0.0.0.0`, which would time out any SSH session. It is possible to write a script to automate this and not loose connectivity with the host but this is not recommended for the inexperienced or faint of heart.

Preparation

If we want to allow users to run UML sessions that attach themselves to tun devices on the host they need access to `/dev/net/tun`. One way of allowing this is to add a group called "uml" and put the users that run UML sessions into this group. A typical `/etc/group` entry might look like `"uml:x:104:david"`, or you may wish to use your distributions **`addgroup`** or **`groupadd`** command.

Now that we have created the group we need to make sure `/dev/net/tun` is owned by the group "uml" and that the group has access to it.

```
# chown root:uml /dev/net/tun
# chmod 660 /dev/net/tun
```

Creating a bridge

The following sets up a bridge, a tun device and attaches the device to the bridge. We must do this before we start our UML sessions. The steps are explained in detail below and it is recommended you read through the descriptions before trying it out.

```
 1 - # ifconfig eth0 0.0.0.0 promisc up

 2 - # brctl addbr uml-bridge
 3 - # brctl setfd uml-bridge 0
 4 - # brctl sethello uml-bridge 0
 5 - # brctl stp uml-bridge off
 6 - # ifconfig uml-bridge 192.168.0.100 netmask 255.255.255.0 up

 7 - # brctl addif uml-bridge eth0

 8 - # tunctl -u umluser -t uml-conn0
 9 - # ifconfig uml-conn0 0.0.0.0 promisc up

10 - # brctl addif uml-bridge uml-conn0
```

1. First we configure what was our external interface to have an IP of `0.0.0.0`, to be promiscuous and to be up. Devices that are attached to bridges need to have an IP address of `0.0.0.0`.

Warning

Please remember that if you configure your external interface with the IP address 0.0.0.0 you will loose connectivity with it.

2. Now we make our bridge and call it `uml-bridge`. This creates a bridge with nothing connected to it.
3. This sets the forwarding delay, the time the bridge spends in the "listening" and "learning" states, to zero seconds.
4. How often we send out STP "hello" packets with information on the bridged topology. This isn't strictly necessary, due to (5).
5. Turn off STP, the "spanning tree protocol". We only have one bridge and therefore don't need this on. See the Linux-Bridge-STP howto for more information on it.
6. Set the IP and netmask of the bridge. This is really the IP that we want the host machine to have as (unmanaged) transparent devices don't have IPs. The bridge should now appear in the output of **`ifconfig`**.
7. Add the eth0 device to the bridge. Provided you set the IP of the bridge in step (6) you should now see network connectivity back on the host. Test this by attempting to ping another device on the network, connecting to something (etc).
8. Create a tun device, owned by the user `umluser` and called "uml-conn0". The user `umluser` has to exist on the host.
9. Configure the tun device to have the IP `0.0.0.0`. We do not set the desired IP for the UML here, that is done inside the UML itself. As with the bridge, the tun device should appear in the output of **`ifconfig`** once we've finished.
10. Add the tun device to the bridge.

This gives us a bridge and a tun device. We connect eth0 to the bridge so that the host has networking too. If we wanted to run more than one UML session on the host we would repeat steps 8 to 10, changing the name of the tun device to something unique each time.

The following (appalling ASCII art) diagram should hopefully give a clearer picture of what we now have.

```
Outside world  --> [      BRIDGE       ]
                   [ Port 0: eth0      ] -->  Networking on the host
                   [ Port 1: uml-conn0 ] -->  Our UML session
                   [ ...               ]
                   [ Port X: uml-connX ] -->  More UML sessions
```

At this stage there is nothing actually connected to the "other end" of the tun device that's on port 1 (uml-conn0). Imagine it as a piece of wire, we've plugged one end into the bridge and the other end gets "plugged in" to the UML when we start it up.

Starting a UML session

Now we need to start a UML session and connect it to our tap device. The following line is merely an example and will most likely not work "out of the box". The important thing to notice is that "eth0" is connected to a tuntap device. Anything after the first comma is considered extra parameters, for tuntap we need only specify the name of the tun device on the host we wish to connect to.

```
# ./linux ubd0=./root_fs mem=64M con=null con0=fd:0,fd:1 \
devfs=nomount eth0=tuntap,uml-conn0
```

Presuming we've now managed to get into the UML session the next step is to login as the superuser and configure the devices. For the network in our example we want to configure our UML with an IP of `192.168.0.X` and a subnet mask of `255.255.255.0`.

```
# ifconfig eth0 192.168.0.201 netmask 255.255.255.0 up
```

We should now be able to communicate with other machines on the local network! Test this by pinging other hosts on the network, connecting to services etc. On the network in the example our gateway to networks other than our own (the Internet, for example) is `192.168.0.100`, which can be set as follows.

```
# route add default gw 192.168.0.100
```

Note

We do not need to set the gateway in order for our UML to communicate with other machines on the network. It just happens that in this example our gateway to the internet is the same host that is running the UML session.

Important

Don't forget to create `/etc/resolv.conf` on the UML filesystem with nameserver information. It should be sufficient to copy this from the host. See **`man resolv.conf`** for more information.

We should now have a host with a bridge running on it that sends traffic around to our UML sessions. If there is a gateway and it is set inside the UML session we should also be able to access the internet.

Removing the bridge

If we want to remove the bridge we first have to reverse what we did to create it. Firstly there should be nothing attached to the bridge when we delete it. Also, the bridge must not have any IP addresses assigned to it, or we wont actually be able to delete it at all. We then need to delete the tun device we created, and put the correct network settings back for eth0.

```
# ifconfig uml-bridge down

# brctl delif uml-bridge eth0
# brctl delif uml-bridge uml-conn0
# brctl delbr uml-bridge

# tunctl -d uml-conn0
# ifconfig eth0 192.168.0.100 netmask 255.255.255.0 up
```

Chapter 6. Frequently asked questions

Q:
I run a UML hosting provider, how can I restrict what IP addresses a UML can use?

A:
This can be done using "ebtables", a URL for this is available at the end of the document. In newer 2.6 kernels, iptables can also accomplish this.

Q:
Why can multiple UMLs on a bridge talk to the host but not each other?

A:
Often this is caused by UML instances having the same MAC address. Please see the next question.

Q:
Why do all my interfaces have the same MAC address?

A:
This could due to the way interfaces are brought up. If an interface is brought up using **`ifconfig eth0 up`** before it is given an address, the interface will receive a "default" MAC address. Differences between scripts on various distributions cause this error to surface. You can either edit the scripts on your distribution to bring up interfaces correctly, or see the following question.

Q:
How can I guarantee a UML network interface will always have the same MAC address?

A:
A unique MAC address can be assigned by specifying it as part of the network interface argument, like so: **`eth0=tuntap,uml-conn0,FE:FD:00:00:00:01`**.

Q:
If I have daemons on the host machine that bind to all addresses, what will they actually bind to?

A:
The daemons will (normally) bind to any IP that isn't `0.0.0.0`. This means they wont bind to addresses in use by your UML sessions, just the IP used by the bridge, localhost and any other network adapters you happen to have.

Q:
Can I add more than one physical adapter to the bridge?

A:
Of course, bridges are designed for this. Simply repeat steps (1) and (7) in Chapter 2, *The host kernel*.

Q:
My ethernet adapter has more than one IP, how can I make this work with bridging?

A:
Simply add aliases to the bridge interface in the same way you would any other interface.

Q:
I get the error `"br_add_bridge: Package not installed"` when trying to add a bridge, why is that?

A:

This happens if you don't have bridging support compiled into your kernel, or the module isn't loaded. Read the section about the host kernel.

Q:
I get the error `"Failed to open '/dev/net/tun' : No such device"`, what could be wrong?

A:
Most likely you don't have tuntap compiled into your kernel, or the module isn't loaded. Read Chapter 2, *The host kernel*.

A:
If your host kernel uses devfs, the TUN device can also be `/dev/misc/net/tun`. Add the following lines to `/etc/devfsd.conf`:

```
REGISTER       misc         MKOLDCOMPAT

# Create correct /dev/net/tun symlink
REGISTER       ^misc/net/tun$  CFUNCTION GLOBAL unlink   net/tun
REGISTER       ^misc/net/tun$  CFUNCTION GLOBAL symlink  /dev/$devname
net/tun
```

and comment out the old UNREGISTER for misc if it exists:

```
#UNREGISTER     misc         RMOLDCOMPAT
```

then send a HUP signal to the devfsd process to force it to read the modified configuration file. This will create a symlink `/dev/net/tun` pointing to your actual TUN device `/dev/misc/net/tun`.

Q:
I get the error `"Failed to open /dev/net/tun, err = 13"`, what could be wrong?

A:
Most likely, permissions on `/dev/net/tun` on the host are incorrect. If you are not running UML as root, add group permissions to `/dev/net/tun` and put the user who runs UML into the appropriate group.

Q:
I get various errors about the network being unreachable on the host after I've added the bridge, what could be the cause?

A:
Check that you assigned the bridge an IP address and that it is up. The output of **`ifconfig`** will help here.

A:
Setting up the bridge may remove your default route, in which case you need to issue the command **`route add default gw <ip>`** on the host, where <ip> is the IP address of your gateway.

Q:
Does ARP travel across my bridge?

A:
Yes, ARP will travel across bridges, this is necessary in order for ethernet to work as it should. If

you are not seeing ARP pass your bridge interface this may be an indication of a problem.

Q:
Can I use DHCP with bridging?

A:
DHCP is fully compatible with bridging as described in this HOWTO. Setup a DHCP server somewhere on the network and use dhclient or dhcpcd in the UML.

Q:
I seem to be having trouble with Samba and Windows SMB clients when using UML, is there a fix?

A:
Configuring a Samba server on your LAN to act as a WINS replicator fixes these issues in many cases. See this page [http://samba.mirror.ac.uk/samba/docs/man/howto/NetworkBrowsing.html] at the Samba site for more information on WINS.

Appendix A. References and credits

I've used a number of other documents extensively to create this one. The first is David Coulsons "pseudo-dedicated hosting service" page, another is the Linux-Bridge-STP howto. The URLs for both of these are available below.

I received some very useful help in the initial stages of this HOWTO from Christopher S. Aker. Thanks also to Dr. James Walden for his correspondence and additions to the FAQ section.

The icons used in this document were created by Jakub Steiner. They are available at his website [http://jimmac.musichall.cz/i.php3?ikony=83] and are used with permission.

Useful URLs

- The official UML page [http://user-mode-linux.sourceforge.net/]
- Linux bridging page [http://bridge.sourceforge.net/]
- Linux-Bridge-STP HOWTO [http://www.tldp.org/HOWTO/BRIDGE-STP-HOWTO/index.html]
- David Coulson's networking page [http://uml.openconsultancy.com/#network]
- ebtables [http://ebtables.sourceforge.net/] / ebtables relation to bridging [http://ebtables.sourceforge.net/br_fw_ia/br_fw_ia.html]