Ethernet Bridging With OpenVPN

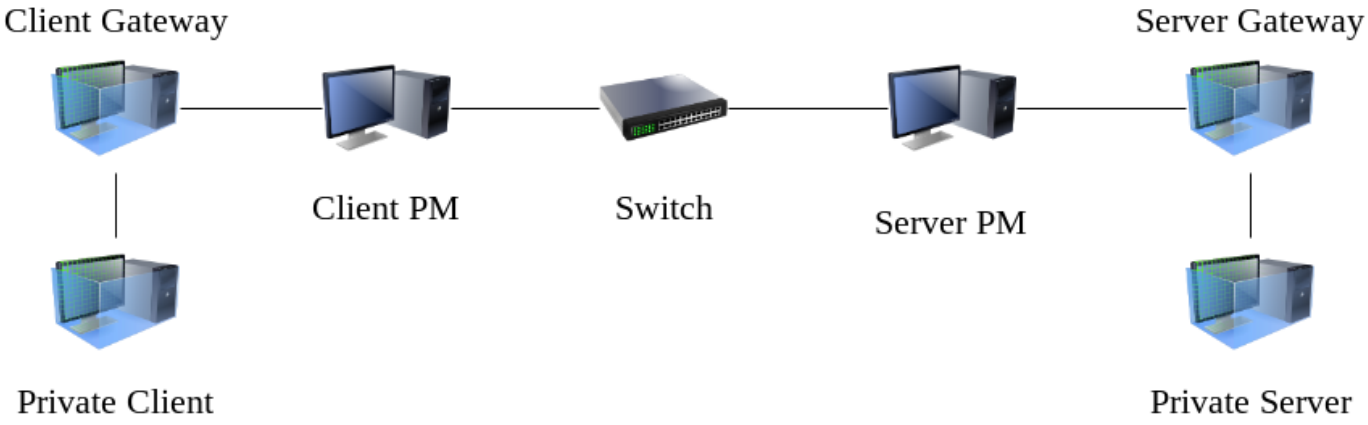
Other guides detail setting up of OpenVPN for 'tun' bridging, where IP traffic is efficiently routed between a couple of geographically separated sites, but this guide is about so-called 'tap' bridging. In effect, it describes how to join a couple of sites forwarding all ethernet traffic between, regardless of protocol. This is useful for development, test networks and if you need to forward non-IP protocols. It is also conceptually simpler than tun bridging because you don't have to care about routing, and you only need run a single DHCP server on one or other side of the bridge.

About OpenVPN

OpenVPN is covered in another article, so I will reference sections in that rather than just copy-paste parts of it here. This will hopefully plug the gaps required to get a working ethernet bridge.

Introduction

Just so we are clear what we're talking about, here is a diagram:



There are two physical machines (Client PM and Server PM) and four virtual machines. The aim is that the 'Private Client' talks directly to the 'Private Server', as though they are on the same ethernet segment. The switch in the middle is just my home network but you could consider that to be the Internet if you like.

Many real setups will make the 'Server Gateway' and the 'Private Server' one and the same, there is no reason for them to be separate machines, however I felt it's easier to follow what's going on if they are distinct.

VirtualBox

VirtualBox is my preferred virtualisation solution because it installs cleanly under Slackware 14.2 and -current and is pretty easy to use. If you happen to have four physical machines on which you can

install Slackware you can of course do all of this without virtualisation! In my case I have two physical machines to represent each 'site' that you want to bridge together. All virtual machines run un-patched Slackware64 14.2

Private Server

Without further ado, let's get on with setting up our private server. I've given the virtual machine the following resources:

- 1024 MB RAM
- 8GB Hard disk

In the VirtualBox network settings, I've told the virtual machine to use 'Internal Network' and I've typed in the network name 'slacknet' for want of a better name. You could call it 'private' if you like. Nothing will reach this network unless we setup a machine as some kind of router to it.

I've told Slackware to install the A, AP, N and L disk sets because I don't need any GUI but want networking stuff, but it doesn't really matter, make it a full install if it pleases you. I've taken care to give this machine a static IP address, in this case 10.0.0.1. Otherwise the install is following all defaults.

After boot there is just one thing we need to do to make this test realistic. We need to enable Dnsmasq. You can follow that link for details, but we just need a small configuration file /etc/dnsmasq.conf containing only:

```
interface=eth0
dhcp-range=10.0.0.50,10.0.0.70,12h
```

That's it! Backup the old dnsmasq.conf somewhere first if you're worried about losing all the comments in it, but it doesn't matter, this is really just a throw-away test machine. The above is all you need for working dhcp server. That's what makes Dnsmasq so great.

Now we just need to start the server:

```
# chmod 755 /etc/rc.d/rc.dnsmasq
# /etc/rc.d/rc.dnsmasq start
```

So now we have our private server running on our private network! Nothing can see it for the moment but if you were to install another virtual machine and connect it to that private (slacknet) network it would get an IP address. We have done this purely to demonstrate that non-IP (i.e. DHCP) traffic can get across our bridge and DHCP is just a convenient way of doing this.

Server Gateway

VirtualBox VM Configuration

The server gateway is another Slackware machine setup with the same memory, hard disk and disk sets as the 'Private Server' above. It differs, however in that it has a second NIC configured in VirtualBox.

The first NIC in VirtualBox aka 'Adapter 1' in the VirtualBox GUI will end up being eth0 in the booted Slackware virtual machine, just bear this in mind. I will set this as 'Bridged Adapter' under 'Attached to:' in the GUI. Note that for all NICs I'm setting 'Promiscuous Mode' to 'Allow All', although I've no idea if that's needed for all the NICs on all the machines in this guide.

'Adapter 2' will be connected to the 'slacknet' private network discussed in the 'Private Server' configuration above.

Network setup

Now all the questions Slackware asks you during network setup apply to the public interface which is connected directly (via VirtualBox's bridge) to the switch in the middle of the diagram above. Chances are it is your home broadband router which will dish out an IP address and maybe (if you're lucky) an appropriate DNS name. It matters not, so long as there is something you can use to identify this public interface on the network. From here on I'm going to refer to this machine's ethernet interface as 'vpn', or we can say 'vpn.localnet', however if you only have an IP address just use that. The good news is that it doesn't matter if it 'clashes' with the IP address range we talked about in 'Private Server' dnsmasq configuration because... well... it's really private. Neat huh? This would obviously be a big issue if we want to later add routing from 'Private Server' to the outside world, but that will be another howto.

tap0 device

Having installed a nice shiny 2nd Slackware virtual machine with two ethernet cards, only one of which is active we need to do some extra stuff before we talk about OpenVPN.

The first step is to install tunctl. I'm going to make the (possibly unfair) assumption that you know how to install Slackware packages from slackbuilds.

Once tunctl is installed, edit /etc/rc.d/rc.inet1, find the following lines:

```
# Function to start the network:
start() {
  lo_up
...
```

Make them say:

```
# Function to start the network:
start() {
  lo_up
  /usr/sbin/tunctl -u nobody -t tap0
```

```
...
```

This will (on boot) create a tap0 device with the correct permissions such that OpenVPN can access it and, more to the point, it will create it before any of the bridging configuration stuff in the Slackware init scripts kicks in. rc.local would have it created way too late.

Bridging setup

OK, so now we are talking about kernel network bridging, not the ethernet bridging that's the subject of this article. It's a sad fact that even though we have a NIC purposely allocated to the task of talking to our private network, i.e. not shared with any other traffic, we still need to create a kernel bridge in order for OpenVPN to talk properly to the NIC, in other words we can't tell OpenVPN to just 'use eth1' (oh that it were so simple). It wants a 'tap'. And a tap can only talk to a NIC via a bridge. So let's have a look at the network config in Slackware.

For eth0 in /etc/rc.d/rc.inet1.conf you are going to now have a setup something like this, courteously supplied by the Slackware netconfig script:

```
IPADDR[0]=""
NETMASK[0]=""
USE_DHCP[0]="yes"
...
```

Or you may have a static IP address. You should have *something*. Now scroll down to the section on bridging:

```
# Example of how to configure a bridge:
# Note the added "BRNICS" variable which contains a space-separated list
# of the physical network interfaces you want to add to the bridge.
...
```

Just below that you want the following to appear:

```
IFNAME[1]="br0"
BRNICS[1]="eth1 tap0"
IPADDR[1]="0.0.0.0"
```

Be careful with the numbers. It's the the square brackets which are telling Slackware this is the 2nd device we're configuring as a bridge, and we want to add eth1 to the bridge along with the tap0 device. Slackware doesn't configure the eth1 device, we don't even need to bring it up or otherwise refer to it in the config files as it's slave to the bridge.

Some people might configure br0 as br1 here, to indicate it's connected to the second NIC and keep them the same. You can do that. And you can change the name of tap0 as well if you want, experience tells me it's best to always use the lowest numbers for everything regardless of what they do. For me, at least, it reduces the confusion.

You don't have to set the IP address to 0.0.0.0, but it's a handy way of making the bridge come 'up' without giving it an IP address, and something that is supposed to simply sit there forwarding ethernet frames has no business having an IP address of its own. If you don't set any address at all you can do 'ifconfig br0 up' at some later point.

When you boot with this configuration, you should check that br0 is up with

```
# ifconfig br0
```

And you can also check the bridge with 'brctl show br0' and it should show something like this:

```
# brctl show br0
bridge name bridge id        STP enabled    interfaces
br0     8000.485b39c042ee    no         eth0
                                                  tap0
```

So that's done the bridging config. Now OpenVPN (when it's eventually configured) will talk nicely to the private network aka 'slacknet' without any trouble, even if it only has 'nobody' as a username.

OpenVPN setup

So this is the part where I can just rest easy 'on the shoulders of giants' so to speak. Please setup your the PKI stuff for an OpenVPN server by following the guide in the OpenVPN howto. Follow all of section 5 so you have a set of public and private keys for the server. Don't worry about creating the config file to reference them from though as we'll do that back here.

Ready? Good.

So my /etc/openvpn/server.conf is a little simpler than the one in the OpenVPN howto. We don't care about any of the routing stuff, and I've removed all the comments for brevity. You can always check options in the manual page for OpenVPN if you want but most are self-explanatory.

```
port 443
proto udp      # or tcp
dev tap0     # or tun
ca /etc/openvpn/certs/ca.crt
cert /etc/openvpn/certs/vpn.crt
key /etc/openvpn/keys/vpn.key
dh /etc/openvpn/certs/dh2048.pem
tls-server
keepalive 10 120
daemon
log-append  /var/log/openvpn.log
verb 3
```

I have removed the passphrase from vpn.key (remember that's the equivalent of 'server1.key' in the OpenVPN howto). I've avoided giving up root as well. You would run a more secure setup if this was a production system by understanding the options in the OpenVPN manual page, this is just to get you working.

Now you can go back to the other article and look at Setting up the server. Skip forward to the stuff that talks about rc.openvpn, to ensure you have a script to start OpenVPN up. Once you have that you should be able to startup without seeing errors in /var/log/openvpn.log.

Client Gateway

VirtualBox configuration

Like the 'Private Gateway' machine, this machine will have two NICs. 'Adapter 1', however can be NAT instead of bridged, as nothing needs to be able to find it. 'Adapter 2' will be connected to a private 'slacknet' network. I made the name the same as on the other server machine, as they will become connected together but it doesn't matter what you call it so long as 'Private Client' uses the same name.

Slackware Install

I'm afraid once again, you'll need to install tunctl, create a tap0 device and enable the bridging exactly as happened for the Server Gateway. Once tap0, and br0 are created on boot you will be good to configure OpenVPN.

OpenVPN configuration

Once again, I bring into play the other howto for the PKI config on the client side. However you can skip the rc.openvpn creation as we won't be running as a daemon. The client configuration (/etc/openvpn/client.conf) is a little simpler:

```
client
dev tap0
proto udp
remote vpn.localnet 443
ca /etc/openvpn/certs/ca.crt
cert /etc/openvpn/certs/client.crt
key /etc/openvpn/keys/client.key
dh /etc/openvpn/certs/dh2048.pem
remote-cert-tls server
verb 3
```

You should now be able to connect to the 'Server Gateway' with this command as root:

```
openvpn /etc/openvpn/client.conf
```

Private Client

Very much like the 'Private Server' this machine will have similar VirtualBox setup. It will have a single NIC connected to a network called 'slacknet'.

I was going to use Slackware for this, but I decided to cheat and use Slax (Bootable CD distro) to save time on the install. Slax requests a DHCP address on boot, so it really has all you need to test the bridge out (DHCP request sent, DHCP response received). If all goes well when you boot you'll get an IP address something like 10.0.0.50 (or at least in the range of addresses provided by the Dnsmasq server we setup on 'Private Server')

End to End Testing

So once all four (yes!) virtual machines are up and running, it should be possible (if all is working) to boot the Private Client and get an IP address in the right range, i.e. between 10.0.0.50 and 10.0.0.70. Pay close attention to /var/logs/openvpn.log on 'Server Gateway' and of course the messages displayed when running openvpn on 'Client Gateway' as they should tell you what's wrong. OpenVPN is pretty good like that. The paranoid will also want to check /var/state/dnsmasq/dnsmasq.leases on 'Private Server' to reassure themselves that all's well and we are actually talking across the bridge.

Consolidation

As I explained in the introduction, it's possible to test all this out without the 'Private server', and combine the function of 'Private Server' and 'Server gateway' in a single machine. Read on to discover how.

First thing to do is switch off (power down) the 'Private Server'.

Next, we need to assign an ip address to our ethernet bridge on the 'Server Gateway'.

```
# ifconfig br0 10.0.0.1 netmask 255.255.255.0
```

we assign an address to the bridge, not eth1!

Now we just need to run a dnsmasq instance on this br0 interface, to serve requests up to the VPN. Much like the 'Private Server' config, /etc/dnsmasq.conf should look like this:

```
interface=br0
dhcp-range=10.0.0.50,10.0.0.70,12h
```

Then it's just a matter of running up dnsmasq:

```
# chmod 755 /etc/rc.d/rc.dnsmasq
# /etc/rc.d/rc.dnsmasq start
```

At this point, rebooting the 'Private Client' should now get an IP address from the consolidated server, and that's really it. However, I also changed the **old** 'Private Server' to dhcp to confirm it could also get an IP address from 'Server Gateway'

Sources

- Originally written by User bifferos

howtos, ethernet, bridging, author bifferos