VPN PPP−SSH Mini−HOWTO

Scott Bronson

bronson@trestle.com

2001−07−29

Revision History

Revision v1.00	2002−01−16	Revised by: sb
Initial (public) release.		

A PPP−SSH VPN is probably the easiest type of VPN to set up. It uses nothing more than the very common PPP and SSH utilities to form an encrypted network tunnel between two hosts.

Table of Contents

1. Introduction

The technique described in this HOWTO uses PPP to convert packets into a character stream and SSH to encrypt it and transmit it to the remote computer. Most system administrators are well acquainted with the tools and configuration files needed to set up a PPP−SSH VPN.

While it works well with moderate loads over a reliable connection, be warned that a PPP−SSH VPN is subject to some scalability problems. I've included a list of benefits in Section 2.1 and drawbacks in Section 2.2 so you can decide for yourself if a PPP−SSH VPN is a good fit for your needs.

1.1. Copyright

1.2. Disclaimer

You use the information in this document entirely at your own risk. I especially make no guarantees as to the legality or cryptographic strength of the techniques described here. If you feel that you cannot take full responsibility for your setup, then you need to put down this HOWTO and hire one of the many excellent companies who provide accountable, professional VPN service.

1.3. Credits

I took some notes as I adapted Bart Trojanowski's excellent instructions to a newer version of PPP running on my Debian system. A few weeks later, I converted the notes into SGML. Eventually, those evolved into this HOWTO.

Bart's instructions were based on Arpad Magosanyi's good but now fairly dated VPN Mini−HOWTO. If you run into troubles and my document doesn't seem to help, or if you're running an older version of the Linux kernel or PPP, you'll definitely want to give his HOWTO a read.

2. Introduction

2.1. PPP−SSH Benefits

There are a number of benefits to setting up a PPP−SSH VPN. It's relatively simple, it uses common off−the−shelf tools, and it probably won't require a reboot before bringing up the link. Here's a more comprehensive list:

Easy to install

You probably won't need to patch or recompile your kernel, run LILO, reboot, or perform any other perilous administration activities. PPP and SSH are included with most distributions, and most kernels come preconfigured to use them properly.

Easy to set up

You should not have to edit any existing configuration files. You simply customize the script file provided later in this document, which contains all the VPN configuration info, and then execute it on the client machine. Any existing PPP or SSH configurations should continue to work just fine.

No mucking with firewalling

If the SSH protocol currently traverses your firewall, then PPP over SSH will traverse your firewall as well. (If you aren't using SSH, then why not? It is almost a required tool for system administrators nowadays.)

No mucking with manual routing

pppd automatically sets up routing for you. And, if you have very complex routing needs, it's very easy to put the custom routing commands in the script file.

No need for static IP addresses

PPP−SSH VPNs have no trouble whatsoever with dynamic IP addressess. The client must be able to find the server to connect to, of course, but dynamic DNS would work fine for that. Setting up a VPN over a dialup connection is no problem.

Multiple Tunnels are Easy

It's easy to set up multiple tunnels to a single computer. You simply need to make sure that the IP address for each tunnel's network interface is distinct.

2.2. PPP−SSH Drawbacks

This type of VPN is not without a few difficulties. Basically, it doesn't run unattended very well. If you're looking for a production−quality VPN that you can set up and forget about, you will proabably find PPP−SSH a little disappointing. Some alternatives are described in Section 2.4.

Trying to maintain a TCP connection

If the SSH TCP connection is broken for any reason, your VPN goes down hard and takes all tunnelled TCP connections with it. If you have a less than reliable link −− say it's difficult to download more than a few tens of megabytes at one go −− you will be re−starting the VPN a lot.

Running IP packets over a TCP stream

The TCP protocol consists of streams layered on top of IP packets. When you *then* run IP packets over the TCP stream (as we're attempting to do), the personality conflict between the two can become very apparent. Mostly, this manifests itself as weird delays, dropouts, and oscillations. Sometimes you'll see problems at load, sometimes with next to no traffic. Short of changing the entire OSI model (ha ha), there's not much that can be done about this.

Tends to be bursty

For some reason, when network load gets high, one tunneled TCP connection tends to get all the bandwidth and the others get ignored. This leads to timeouts and dropped connections. Theoretically, this is fixable.

Can't reliably tell when link is down

Keepalives are small packets sent to tell the machine on the other end that the connection is still up. If the network load gets too high, keepalives will be delayed. The other machine will mistakenly assume the connection has been dropped and take down its end of the link.

Without keepalives, however, there's no way for either machine tell if the link has been dropped. When one machine tries to bring the link back up, if the other machine thinks it already has it up, confusion can reign. Most often this will show up as multiple ppp network devices, duplicate routes, and tunnels that appear to be up but drop every packet. A liberal use of "killall −9 pppd" will usually set things back in order. A more intelligent start script could probably improve this.

Too many simultaneous connections avalanches fast

When I use regular PPP over a 56K modem and Postfix opens 10+ connections to deliver my outgoing mail, everything works well. However, when I try to run this exact traffic over a VPN tunneled over a much faster DSL link, it stalls out. Ping times skyrocket for a spell (2 minutes and beyond), traffic moves at a trickle for a while, then it stops completely. The only way to get packets moving again is to restart the tunnel. I'm not sure if this is a bug or an inherent limitation. Reducing the number of connections that Postfix maintains for outgoing mail fixed this problem for me..

It's high−overhead, high−latency

Ping times over my 57.6 modem connection are normally in the 130−170 ms range. However, ping times for a PPP−SSH VPN running over the same modem connection are in the 300−330 ms range. Turning on PPP compression can help a lot if you're transmitting compressible data. Email is compressible, Vorbis files are not.

2.3. Suggested Reading

VPN FAQ

The VPN FAQ at http://kubarb.phsx.ukans.edu/~tbird/vpn/FAQ.html is a very good resource. It's comprehensive, kept reasonably up−to−date, and not afraid to express an opinion.

Linux Kernel HOWTO

If your kernel doesn't already have PPP and IP Forwarding capability built−in, the Linux Kernel HOWTO will tell you how to recompile your kernel to add it. It will also tell you how to load and unload the PPP kernel modules.

PPP HOWTO

Tells how to install and set up the PPP daemon if your distribution did not automatically install it for you. Also has an excellent section on linking two networks using PPP. That's pretty much what we're doing, except that we're also encrypting it. You can find it at http://www.linuxdoc.org/HOWTO/PPP−HOWTO/index.html.

SSH HOWTO

I wish there were an SSH HOWTO! For now, the documentation that comes with your distribution should be a good start. You might also check the OpenSSH web site.

Networking Documentation

If you're not very familiar with networking, you'll want to scour the Linux Network Administrators Guide. It's an excellent introduction to most of the concepts we'll be using here. You may also find the Linux Networking HOWTO at http://www.linuxdoc.org/HOWTO/Networking−Overview−HOWTO.html to be a useful introduction, especially itse sections on TCP/IP, PPP, and tunneling.

2.4. Alternatives

There are a ton of VPN technologies in the world now. If PPP−SSH doesn't fit all your needs, you might want to check one of the following packages.

ipsec

ipsec describes a set of low−level protocols, ESP and AH, to perform authentication and encryption at the packet level. It also uses a higher−level protocol, IKE, to negotiate connection parameters and exchange encryption keys.

FreeS/WAN is probably the best Linux ipsec implementation today. Although it can be very difficult to set up, especially for those who are not terribly familiar with networking, it is amazingly stable once it is working. You can find out more at the FreeS/WAN home page.

Another good, free ipsec implementation is Cerberus. Unfortunately, the National Institute of Standards and Technology only distributes Cerberus to US or Candadian citizens currently located in either the US or Canada. Therefore, depending on who you are, obtaining Cerberus ranges from moderately difficult to effectively impossible.

PPTP

PPTP (Point−to−Point Tunnelling Protocol) is a Microsoft−developed VPN protocol, described in RFC2637. It is a very common and well−understood technology and has many mature implementations on all commonly−used computer platforms. However PPTP is generally considered to have somewhat weak security.

Probably the best Linux PPTP implementation is PoPToP, found at http://poptop.lineo.com/.

CIPE

CIPE is Olaf Titz's protocol to encapsulate IP traffic over UDP packets. It has both a Linux version and a Windows version. I haven't used it yet, but it is in strong development and looks very promising. For more information, the CIPE−MASQ Mini−HOWTO is a terse but informative read.

3. Software Installation

3.1. Terminology

Because setting up the VPN very much resembles a client−server transaction, I'll borrow from that terminology to give a name to the computer at each end of the tunnel:

Server

> This is the computer that passively waits for incoming VPN connection requests. It runs completely unattended.

Client

> This is the computer that initiates the connection request, asking the Server to bring up the VPN.

3.2. Requirements

- Your kernel must have support for TCP/IP and PPP compiled in. Almost all distributions ship with PPP support straight out of the box. If yours didn't, or if you're using a custom kernel, I'll include a little more detail about this in Section 3.4.
- You must install the pppd daemon. This most likely comes with your distribution. I'm using ppp−2.3.11. Later versions should work just fine, and earlier versions should also work as well as long as they support the "pty" option. You don't need to install chat or any of the other tools designed to work with PPP −− you just need pppd.
- The client machine needs to have the the ssh client installed. There are many different versions of ssh floating around, but they should all work. I'm using OpenBSD's OpenSSH package version 2.2.0p1.
- The server machine needs the sshd daemon to field ssh requests from the client. OpenSSH includes a very good ssh daemon.
- Finally, you want to install the sudo tool on the server. You can find out more about sudo in the Linux Administrator's Security Guide, in the Administration Tools section, and the Linux Security HOWTO, in the Root Security section. You might also want to look at sudo's home page.

3.3. Planning

To set up a PPP−SSH link, you need to specify the the following parameters:

Server Hostname

> What is the hostname or IP address for your VPN server?

Server VPN Username

> On your server, what username will the VPN software run under? This HOWTO includes instructions on how to create a user named "vpn" specifically for this. This should *not* be root! For security and logging, this should be a dedicated account.

Server Interface IP Address

The PPP−SSH VPN sets up dedicated network interfaces on both the client and the server. The interface will be pppN, where N is the number of the first unused ppp interface (i.e. it will be ppp1 if you're already using ppp0 to dial out over your modem).

You will need to specify the IP address for the interface on the server. This address is only visible to the client and server (and to any machines on subnets that the client and server may be forwarding unmasqueraded packets for).

If you don't know what IP address to pick, read chapter 2.2 in the Linux Network Administrators Guide and especially look at Table 2−1. For example, 192.168.0.1 is a good choice.

Client Interface IP Address

You need to select the IP address for the interface on the client. It must, of course, be on the same network as the server's IP address. It must not conflict with any other networks on the client, nor can it be the same as the server's network interface IP address. If you selected 192.168.0.1 for the previous answer, you would probably use 192.168.0.2 here.

My setup looks like this:

```
SERVER_HOSTNAME = eldivino.domain.com
SERVER_USERNAME = vpn
SERVER_IFIPADDR = 192.168.3.1
CLIENT_IFIPADDR = 192.168.3.2
```

3.4. Set Up PPP

The kernel's PPP code can either be compiled into the kernel itself or it can be put in loadable kernel modules. If you compiled it into the kernel, you can skip on to the next step −− you're done here. However, if you're loading PPP as modules, you need to make sure the modules get properly loaded.

You can check to see if ppp is listed, along with all other currently loaded modules, when you run lsmod. Remember to check that the PPP module is loaded on both the client and the server.

```
server$ /sbin/lsmod
Module                  Size  Used by
ppp                    20780   0 (unused)
slhc                    4376   0 [ppp]
3c59x                  21636   1

client$ lsmod
Module                  Size  Used by
ppp_deflate            40308   0  (autoclean)
bsd_comp                4076   0  (autoclean)
ppp                    20844   2  [ppp_deflate bsd_comp]
slhc                    4376   1  [ppp]
```

If you're sure ppp was compiled as a module, but it's not loaded into the kernel, try loading it with modprobe:

```
# modprobe ppp
```

If modprobe didn't return any errors, check lsmod again −− it should be listed this time. If so, then your ppp module is not being loaded at boot time. This is OK if you're running the kernel daemon as the PPP modules will be loaded on demand. If you're not, however, you'll need to manually load the modules at boot time by putting a single line consisting entirely of the word "ppp" in your /etc/modules file.

See the Linux Kernel HOWTO for more on this subject.

3.5. Allow SSH Through the Firewall

The only network traffic between the two machines (as a result of the tunnel, of course) will be over the SSH protocol.

SSH uses only TCP streams −− no UDP or ICMP. The ssh server (sshd) listens on port 22. Our client (because we use the −P flag) only uses the unpriveleged ports from 1024 through 65535. This description should have given you enough information to set up your firewall.

For example, here are the ipchains commands needed to allow ssh connections to the server. We allow incoming SSH connection between port 22 on the local machine and any port on the remote. Replace eth0 with the interface that will be carrying the ssh traffic and $IPADDR with the IP address of that interface.

```
ipchains −A input  −i eth0 −p tcp −d $IPADDR 22 −j ACCEPT
ipchains −A output −i eth0 −p tcp ! −y −s $IPADDR 22 −j ACCEPT
```

And, here are the commands needed to set up the firewall on the client. We don't allow incoming ssh connections, and we only allow the protocol to pass between port 22 on the remote machine and unprivileged ports on this machine. Again, replace eth0 with the interface that will be carrying the ssh traffic, and $IPADDR with the IP address of that interface..

```
ipchains −A input  −i eth0 −p tcp ! −y −−source−port 22 −d $IPADDR 1024:65535 −j ACCEPT
ipchains −A output −i eth0 −p tcp −s $IPADDR 1024:65535 −−destination−port 22 −j ACCEPT
```

4. Configure the Server

We need to set up the server to respond to the client's request to bring up the tunnel.

4.1. Create a VPN User

The incoming SSH VPN requests must be directed to a particular user on the server. For security and accountability, I recommend you use a dedicated user to field VPN requests. The following steps will set up a system user named "vpn" to do just that.

1. First, we create the user's account. Accounts come in two ranges: the system range (typically 100−999) and the regular user range (1000+). "−−system" tells adduser to add the user in the system range and to give him /bin/false for the login shell. "−−group" tells adduser to also create a group of the same name as the user, and to add the user to the group.

   ```
   server# adduser --sytem --group vpn
   ```
2. Since the vpn user needs to log in via ssh, change vpn's shell from /bin/false to /bin/bash in the /etc/passwd file. You can simply edit /etc/passwd using vi or any other decent text editor.
3. Create a password for the vpn user. It can (and should) be very complex, since you'll only type it a few times while setting up the VPN. After that, you'll never type it again.

   ```
   server# passwd vpn
   Enter new UNIX password:
   Retype new UNIX password:
   passwd: password updated successfully
   ```
4. Now, try connecting to the server to ensure that you've created the account properly.

   ```
   client% ssh eldivino.domain.com -l vpn
   vpn@eldivino's password:
   Linux eldivino 2.2.19 #6 Mon Jun 4 10:32:19 PDT 2001 i686 unknown
   No mail.
   vpn@eldivino:~$
   ```

   It may take a while for ssh to connect if you don't have reverse DNS set up properly. You can fix that whenever you want. It will only delay bringing up the VPN −− it won't prevent it from working.

   If it just stalls, then the ssh protocol is probably being dropped by a firewall between the two machines. Have a look at section Section 3.5 again.

4.2. Set up Authenticated Login

It would be terrible to have to type in a password every time you wanted to bring the VPN link up, so we'll set up SSH's RSA authentication. Skip this section if you truly don't mind typing a password every time.

1. Ensure that the root account on the client machine has a public key in root's home directory (~/root/.ssh/identity.pub). If this file doesn't exist, then you must create it. As root, run ssh−keygen:

   ```
   # ssh-keygen
   Generating public/private rsa1 key pair.
   Enter file in which to save the key (/root/.ssh/identity):
   Enter passphrase (empty for no passphrase):
   Enter same passphrase again:
   ```

```
Your identification has been saved in /root/.ssh/identity.
Your public key has been saved in /root/.ssh/identity.pub.
The key fingerprint is:
15:61:57:7e:5c:26:91:09:5c:e6:10:b7:a1:74:bd:25 root@paradis
```

2. Now, copy identity.pub to the vpn account's authorized_keys file on the server. You will almost certainly have to create this. As root, perform the following commands on the server:

   ```
   server# cd ~vpn
   server# mkdir .ssh
   server# chown root.vpn .ssh
   server# chmod 755 .ssh
   server# cd .ssh
   ```

   Now, copy th the client's /root/.ssh/identity.pub file (it's only one line) to the server's ~vpn/.ssh/authorized_keys file. You can add more lines to authorized_keys, one for each client, if you want to allow multiple clients to connect.

   ```
   server# chown root.vpn authorized_keys
   server# chmod 644 authorized_keys
   ```
3. Now, become root on the client, and try SSHing to the server. You may or may not need to use the −P option, depending on how your client's firewall is set up. If port 22 is blocked on your client (not a bad idea if it's not running an SSH server), then −P tells ssh to use an unprivileged port even though it's running as a priveleged user.

   ```
   client# ssh −P eldivino.domain.com −l vpn
   Linux eldivino 2.2.19 #6 Mon Jun 4 11:03:22 PDT 2001 i686 unknown
   No mail.
   vpn@eldivino:~$
   ```

   There, we were just RSA−authenticated. Keep your private key (the client's ~root/.ssh/identity file) private! Anyone who has access to this file can log into the VPN account on the server.

4.3. Set Up sudo

pppd needs to run as root. However, on the server, we're running everything as the "vpn" user. How can the vpn user run pppd?

There are a number of ways of solving this problem. One is to use the suid bit, and arrange permissions by groups. However, this can get confusing and difficult to administer pretty fast, leading to unintentional security holes. Personally, I find the sudo utility to be a much better solution.

sudo gives ordinary users superuser powers, but only for a very limited set of commands. The system administrator gets to decide what commands are allowed and how much logging to perform. In this case, we want to allow the user "vpn" to run pppd with superuser privilege, but not be allowed to do anything else.

1. We need to edit sudo's configuration file, /etc/sudoers. To use proper locking, hopefully preventing accidents and race conditions, use the visudo command to edit /etc/sudoers. If you're not faimiliar with vi, see the VIM HOWTO.

   ```
   server# visudo
   ```

   Add these two lines to the bottom of the file:

```
Cmnd_Alias VPN=/usr/sbin/pppd
vpn ALL=NOPASSWD: VPN
```

2. Now, verify that sudo is set up correctly. As the "vpn" user on the server, try running pppd using sudo:

```
server# su − vpn
server$ sudo /usr/sbin/pppd noauth
~9}#ÀZ}!}!} }9}"}k} }r} }'}%}zt2−·}'}"}
```

If you get a whole bunch of PPP garbage to the screen (like the last line above), this is good. It means that the vpn user is allowed to run pppd. You can now switch to another terminal to kill it off, or you can just let pppd finish on its own. It should give up trying to connect after 30 seconds or so.

However, if you get "bash: /usr/sbin/pppd: Permission denied" or some other sort of error, or it asks for a password, then sudo is probably not working. You'll need to try figure out what is going wrong. Verify that pppd is in /usr/sbin, and that you set up the sudoers file correctly.

5. Configure the Client

If ppp and ssh are set up on the client, and the server is ready to connect, then all we need to do on the client is create the script to bring up the link.

5.1. Install the Script

The VPN connection is initiated using the vpn−pppssh script below.

1. Save this file on the client (it doesn't matter where −− /usr/local/bin/vpn−pppssh is a good place) and make it executable by running "chmod a+x vpn−pppssh".
2. Fill in the settings at the top of the file with the values you decided on in Section 3.3.

   Remember that this is running under bash so you'll need to avoid whitespace around the equals sign, use quotes where necessary, and escape metacharacters such as $. See the BASH Programming Introduction or Advanced Bash Scripting Guide for more.

   ```
   SERVER_HOSTNAME=eldivino.domain.com
   SERVER_USERNAME=vpn
   SERVER_IFIPADDR=192.168.3.2
   CLIENT_IFIPADDR=192.168.3.1
   ```

   Run "vpn−pppssh config" to print out a list of the configuration variables. This way, you can confirm that your settings are being interpreted correctly.

5.2. The vpn−pppssh Script

Here is vpn−pppssh. All the action is on one line. (the one beginning with "PPPD" in the start clause). All the rest of this file is just support code.

```
#!/bin/sh
# /usr/local/bin/vpn-pppssh
#
# This script initiates a ppp-ssh vpn connection.
# see the VPN PPP-SSH HOWTO on http://www.linuxdoc.org for more information.
#
# revision history:
# 1.6 11-Nov-1996 miquels@cistron.nl
# 1.7 20-Dec-1999 bart@jukie.net
# 2.0 16-May-2001 bronson@trestle.com


#
# You will need to change these variables...
#


# The host name or IP address of the SSH server that we are
# sending the connection request to:
SERVER_HOSTNAME=eldivino.domain.com
```

```
# The username on the VPN server that will run the tunnel.
# For security reasons, this should NOT be root.  (Any user
# that can use PPP can intitiate the connection on the client)
SERVER_USERNAME=vpn

# The VPN network interface on the server should use this address:
SERVER_IFIPADDR=192.168.3.2

# ...and on the client, this address:
CLIENT_IFIPADDR=192.168.3.1


# This tells ssh to use unprivileged high ports, even though it's
# running as root.  This way, you don't have to punch custom holes
# through your firewall.
LOCAL_SSH_OPTS="-P"


#
# The rest of this file should not need to be changed.
#



PATH=/usr/local/sbin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/bin/X11/:

#
# required commands...
#

PPPD=/usr/sbin/pppd
SSH=/usr/bin/ssh

if ! test -f $PPPD  ; then echo "can't find $PPPD";  exit 3; fi
if ! test -f $SSH   ; then echo "can't find $SSH";   exit 4; fi


case "$1" in
  start)
    # echo -n "Starting vpn to $SERVER_HOSTNAME: "
    ${PPPD} updetach noauth passive pty "${SSH} ${LOCAL_SSH_OPTS} ${SERVER_HOSTNAME} -l${SERVER_USERNAME} -o Batchmode=yes sudo ${PPPD} nodetach notty noauth" ipparam vpn ${CLIENT_IFIPADDR}:${SERVER_IFIPADDR}
    # echo "connected."
    ;;

  stop)
        # echo -n "Stopping vpn to $SERVER_HOSTNAME: "
        PID=`ps ax | grep "${SSH} ${LOCAL_SSH_OPTS} ${SERVER_HOSTNAME} -l${SERVER_USERNAME} -o" | grep -v ' passive ' | grep -v 'grep ' | awk '{print $1}'`
        if [ "${PID}" != "" ]; then
          kill $PID
          echo "disconnected."
        else
          echo "Failed to find PID for the connection"
        fi
    ;;

  config)
    echo "SERVER_HOSTNAME=$SERVER_HOSTNAME"
    echo "SERVER_USERNAME=$SERVER_USERNAME"
    echo "SERVER_IFIPADDR=$SERVER_IFIPADDR"
    echo "CLIENT_IFIPADDR=$CLIENT_IFIPADDR"
  ;;
```

```
  *)
    echo "Usage: vpn {start|stop|config}"
    exit 1
    ;;
esac

exit 0
```

6. Bring up the Link

Everything should now be set up. Now it's time to take a deep breath and try to bring up the link.

1. Become root on the client machine and execute the vpn−pppssh script.
   ```
   client# /usr/local/bin/vpn-pppssh start
   ```
2. It will take a while to connect, but then it should come back with something like the following
   ```
   Using interface ppp1
   Connect: ppp1 <--> /dev/pts/1
   local  IP address 192.168.3.1
   remote IP address 192.168.3.2
   ```
3. Did it work? First try pinging the client's VPN interface:
   ```
   client$ ping 192.168.3.2
   ```
4. If this worked, then you can reach the interface on the client OK. Don't get excited yet −− that was the easy part. Now, try pinging the server's VPN interface:
   ```
   client$ ping 192.168.3.1
   ```

   If you get echoes back, then congratulations! Your PPP−SSH VPN appears to be healthy. Packets are successfully travelling the route in both directions. You might want to log into your server and try initiating pings from the server to the client, but at this stage of the game, that's almost guaranteed to work.

You bring the VPN down with "vpn−pppssh stop".

Now that the tunnel works, you might want to integrate it into your system so it comes up automatically as described in Section 7. Also, if you want to forward packets from an entire subnet over the link (rather than just the packets originating on the client and server as we have set up now) see Section 8.

6.1. Troubleshooting

The script itself is fairly simple. The entire system, however, involves a lot of small parts. If any one of them is misconfigured, it can prevent your VPN from working without so much as a message why. Here is a list of things to check if you run into difficulties:

- Double and triple check your network values. Try running "vpn−pppssh config" to ensure the configuration is correct and the shell hasn't ruined any of your values.
- Go back over each step and make sure that it all checks out.
- Try temporarially turning off any firewalls on the client, on the server, and on any machines in between to see if any of them are getting in the way (not likely if you can SSH between the two machines).
- Ensure that your routes are correct. You can list your routes using "route −n". See the Linux Network Administrators Guide and http://www.linuxdoc.org/HOWTO/Adv−Routing−HOWTO.html for more.

6.1.1. sendto: Operation not permitted

When you try to ping the VPN interfaces, if you get a "sendto: Operation not permitted" error, you are probably running into a firewall on the local machine that is denying packets before they even reach the VPN network interface. Your firewall must allow SSH traffic over your regular network *and* it must allow all traffic over your VPN interfacess.

The ipchains commands to smash a hole in your firewall for your PPP interface will something like this:

```
ipchains −I input  1 −i ppp1 −s 192.168.3.0/24 −j ACCEPT
ipchains −I output 1 −i ppp1 −d 192.168.3.0/24 −j ACCEPT
```

ppp1 must, of course, be the network interface of your PPP−SSH VPN, and the IP addresses must match the address of the local interface. Make sure that packets are allowed on both the client and server.

See the Linux Firewall HOWTO, the IPChains HOWTO for kernel 2.2, or documentation on iptables for kernel 2.4.

7. Integrating the VPN into your system

Bringing up the link by hand gets tiring after a while. You probably want your VPN to come up either at boot time or when your dial−up connection comes up.

7.1. Connecting at Boot Time

It's quite easy to get this script to run at boot time. I assume you're using the very common System V initscript setup. If not, you'll have to figure out how to integrate this with your system on your own.

1. Either copy or symlink the vpn−pppssh script to /etc/init.d.
   ```
   cp /usr/local/bin/vpn-pppssh /etc/init.d/vpn-pppssh
   ```
2. Uncomment the echo lines in the start and stop clauses in the vpn−pppssh script to enable the boot−time "Starting" and "done." messages.
3. Put "> /dev/null 2>&1" after the line beginning "${PPPD}" in the start section of the script. This just prevents pppd's verbose messages from mucking up your boot screen. You could also redirect pppd's messages (which may include a very informative error) to a log file or, if you're not aesthetically inclined, leave it alone and let your screen get all mucked up.
4. Now, you simply link your script in to each of the six runlevels.
   ```
   client$ ln -s /etc/init.d/vpn-pppssh /etc/rc0.d/K10vpn-pppssh
   client$ ln -s /etc/init.d/vpn-pppssh /etc/rc1.d/K10vpn-pppssh
   client$ ln -s /etc/init.d/vpn-pppssh /etc/rc2.d/S99vpn-pppssh
   client$ ln -s /etc/init.d/vpn-pppssh /etc/rc3.d/S99vpn-pppssh
   client$ ln -s /etc/init.d/vpn-pppssh /etc/rc4.d/S99vpn-pppssh
   client$ ln -s /etc/init.d/vpn-pppssh /etc/rc5.d/S99vpn-pppssh
   client$ ln -s /etc/init.d/vpn-pppssh /etc/rc6.d/K10vpn-pppssh
   ```

Now, when you reboot your machine, the vpn should come up near the end of the boot process. When it hits this script, your machine will wait until the VPN is up before it continues booting. If this is an issue, you can write your own /etc/init.d/vpn−pppssh script that calls the /usr/local/bin/vpn−pppssh script in the background. The link will come up as your machine finishes booting.

To manually bring the link down or up, just run the vpn−pppssh script directly from /etc/init.d:

```
client$ /etc/init.d/vpn-pppssh stop
client$ /etc/init.d/vpn-pppssh start
```

7.2. Connecting via Dial−Up

If you're dialing into the internet with PPP, you can bring the VPN up every time you bring up the dial−up connection. This is not difficult, but it does require a fairly recent version of pppd, one that supports both the ipparam option, and the ip−up.d and ip−down.d directories.

1. Create the "vpn−up" file in /etc/ppp/ip−up.d:
   ```
   #!/bin/sh

   if [ "$PPP_IPPARAM" = "vpn" ]; then
           # Don't bring up the vpn if we're bringing up the vpn.
   ```

```
        exit 0
fi

/usr/local/bin/vpn start
```

There's a re−entrancy here that the if statement takes care of. If we're bringing up the regular PPP link, we want to bring up the VPN. But, the VPN is a PPP link itself! If we didn't do anything about this, PPP would recursively spawn itself until it ground your machine to a halt.

The secret is the "ipparam vpn" parameter in the vpn−pppssh script. This sets the IPPARAM variable for this invocation to "vpn", which we then check in the startup script. If it's set to vpn, then we know we're in the middle of bringing up the vpn, so we just exit without error. Otherwise, we fire it up.

2. If you want to punch a hole in your firewall for your VPN when you bring it up, you can simply create an /etc/ppp/ip−up.d/vpn−fw file with the following contents. All the shell variables below are supplied by pppd, so you should be able to use this script unmodified.

```
#!/bin/sh

# Punch a hole in the firewall for the VPN

if [ "$PPP_IPPARAM" = "vpn" ]; then
        ipchains -I input 1 -i $PPP_IFACE -s $PPP_REMOTE -d $PPP_LOCAL -j ACCEPT
        ipchains -I output 1 -i $PPP_IFACE -s $PPP_LOCAL -d $PPP_REMOTE -j ACCEPT
fi
```

3. Create the "vpn−down" file in /etc/ppp/ip−down.d/vpn−pppssh:

```
#!/bin/sh

if [ "$PPP_IPPARAM" = "vpn" ]; then
        # Don't bring down the VPN if we're bringing down the vpn.
        exit 0
fi

/usr/local/bin/vpn stop
```

Make sure to make all the scripts above executable (chmod a+x /etc/ppp/ip−up.d/vpn−pppssh). Now, when you bring up your PPP link, the VPN should come up with it. And, when you shut it down, the VPN will disappear. Easy as pie.

8. Forwarding Between Subnets

You only need to read this section if you're trying to connect networks, not just hosts. I assume your host−to host tunnel works so now your client and server computers can exchange pings flawlessly. Now, it's time to allow networks connected to the client and server machines to use the tunnel as well.

8.1. Forwarding

First of all, you need to be sure that you're allowing packets to be forwarded across your interfaces. You turn this on through the proc configuration interface. It's best to do this once at boot time, but you can also put it in the vpn−pppssh script or even create a script in the /etc/init.d/ip−up.d directory (see Section 7.2) if you want.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv4/conf/ppp1❶/forwarding
```

❶ Of course, you need to replace ppp1 with the appropriate VPN interface (the interface associated with SERVER_IFIPADDR or CLIENT_IFIPADDR depending on if you're doing this on the server or the client).

8.2. Gatewaying

On all computers in the subnet, you need to set up the local VPN host as the gateway to all the networks on the other side of the tunnel. This just tells the computers "if you have any packets destined for the opposite side of the VPN, send them to the local VPN host". Note that if the VPN host is already the default gateway for all the computers, then you don't need to worry about this step −− the packets will be forwarded automatically.

In the example below, my VPN host has IP address 192.168.1.1 on the local network. and IP address 192.168.3.2 on the VPN network. The VPN network, containing both the client and server VPN interfaces and all computers on the opposite side of the link, is 192.168.3.0/24. Therefore, on every local computer that I want to be able to send packets through the VPN, I need to run the following command:

```
# route add −net 192.168.3.0 netmask 255.255.255.0 gw 192.168.1.1
```

Now, any packet destined for the 192.168.3.0/24 network on the local machine will be relayed to host 192.168.1.1 on the local network to be forwarded through the VPN.

8.3. Routing

There should be no need to set up custom routing −− pppd tries to do it all for you. However, if you find that pppd doesn't cover your needs, the place for your routing commands is in the vpn−pppssh script itself. To change the routing on the client, simply run route. To change the route on the server, use ssh to send those commands to the server. Here's an example:

```
route add −net $NET1 gw $SERVER_IFIPADDR
```

```
ssh -o Batchmode=yes $SERVER_HOSTNAME -l$SERVER_USERNAME route add -net $NET2 gw $CLIENT_IFIPADDR
```

8.4. Masquerading

You can even set up one or both hosts to masquerade all connections through the VPN tunnel. See the IP Masquerade HOWTO for more.

```
# ipchains -A forward -i ppp1 -s 192.168.0.0/24 -j MASQ
```

8.5. Now try it

That should be all you need to forward packets from subnets connected to the client or server to the opposite machine. May your PPP−SSH VPNs serve you quietly and reliably for many years to come.