IPv4 Masquerading for the Hypothetical Geek

Ben Pfaff <pfaffben@msu.edu>

24 May 2000

Contents

1 Introduction

Hypothetically, if you're a geek, you have any number of computers just sitting around, and you have enough random parts to build another half dozen or so without video or disks. So, *hypothetically*, the thing to do is to network them all together, just for hack value, using a combination of old thinwire Ethernet cards and a 386 with 6 NE2000s acting as a bridge.

Remember, all this is *hypothetical*. I would never do this. I am not a geek.

The only problem is that all these machines will be wanting Internet access. What good is a computer without Internet access, after all? So you call up the *hypothetical* local DSL provider and order 18 *hypothetical* DSL lines and companion IP addresses, or 25 to allow room for expansion.

Ahem! Not even your *hypothetical* geek can pay for 25 *hypothetical* DSL lines, and anyway no DSL provider exists out in the *hypothetical* boonies where the *hypothetical* geek lives.

So, in reality, your *hypothetical* geek is stuck with a 56 kbps[1] modem on a single phone line. How can such a geek make the best of this? The *hypothetical* geek still wants all the machines to have Internet access. He doesn't want to put a modem in all 18 computers, which would *hypothetically* result in the *hypothetical* geek slitting his *hypothetical* wrists[2] on a jagged case fragment anyhow.

The real answer to this *hypothetical* problem of connectivity is *masquerading*, also known as *network address translation* or NAT. Masquerading makes any number of machines on a single network look like a single host with a single network address. More specifically, masquerading for IPv4[3] has been a feature of the Linux kernel since version 1.2.*x* for some value *x*.

This article covers the basics of IP masquerading under Linux 2.0.*x* and 2.2.*x*. For more information on IP masquerading under Linux, please see David Ranch's excellent IP Masquerading HOWTO, available from your local Linux documentation site.

2 Planning

Figure 1 on page 2 shows the architecture of a typical home network set up under IP masquerading. Notice how all the machines in the network are ultimately connected to the Internet through the server in the lower right of the diagram. This is probably the way

[1] Current FCC regulations limit speed to 53 kbps. It's only your crappy phone lines that limit speed to 20 kbps.

[2] Down, not across.

[3] IPv4, or Internet Protocol version 4, is the version used in real life. IPv6, the next version, is used only in pipe dreams^W^Wexperimental settings.

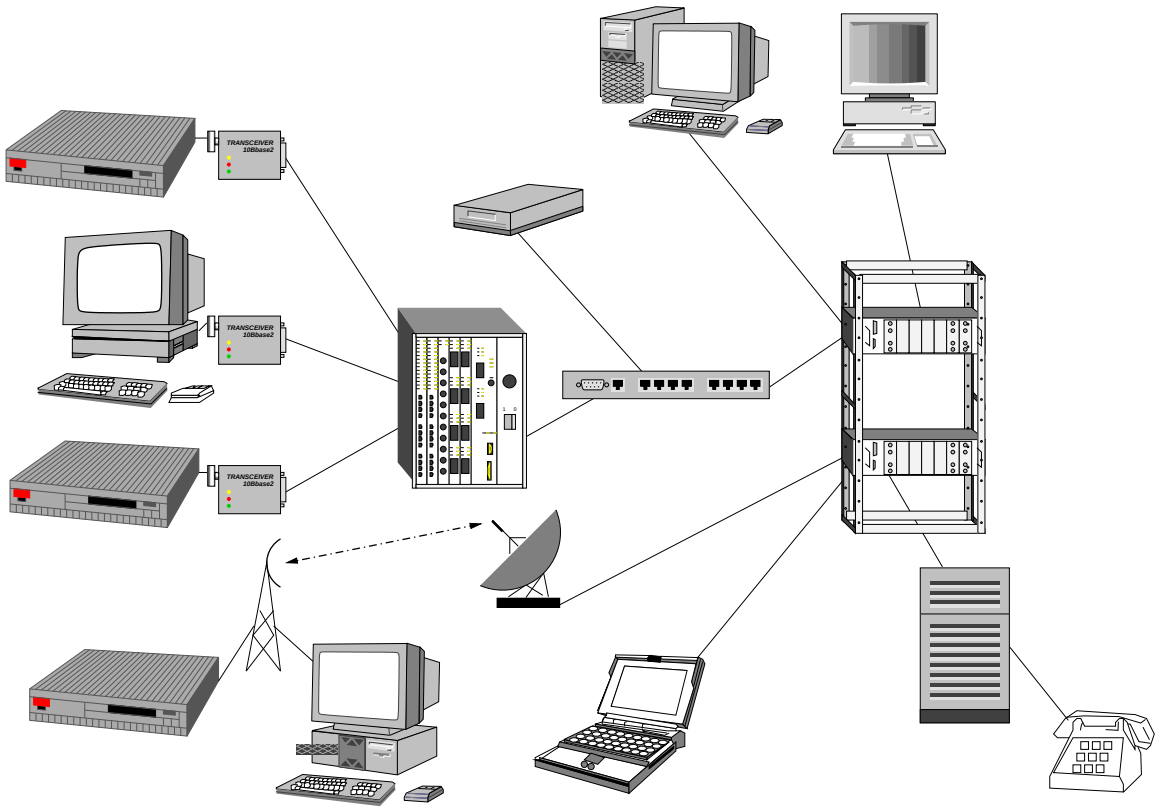

Figure 1: Architecture of typical home network.

that you want to plan for your IP masquerading setup to work.

You'll need all of the following in order to set up masquerading:

- A Linux-capable server computer with a network card and an Internet connection. A 386 or 486 is fine for basic setups.
- One or more client computers with network cards. These can run just about any operating system.
- A hub to connect all the computers together.
- A range of IP addresses (see below).
- A geek, for debugging purposes. This can be the most expensive part of the setup, but some geeks can be bribed with interesting bits of old hardware or discounts on new hardware.

You'll have to come up with the hardware and the geek on your own. But selecting the IP address range to use is easy. I recommend that you use the 16-bit "Class B"-size address range at 192.168.x.y, which is reserved specifically for private networks like the one you're building. As long as you have fewer than 65,000 or so computers on your network[4], this should be plenty.

As a consequence, the examples below assume that all of your computers have IP addresses of the form 192.168.0.x, where x is a number between 1 and 254[5]. Specifically, it is assumed that your server is at 192.168.0.1.

Now that we're done with the formalities, plug everything together and proceed to the next section, where we'll set up basic masquerading on the server and the clients.

[4] Around here, that reads "as long as you're not Ed Glowacki."

[5] 192.168.0.0 and 192.168.0.255 are reserved for the network address and the broadcast address, respectively.

3 Basic server setup

The toughest setup for masquerading is on the server, and even that isn't very hard. Now, let's a look at how to actually do it.

Start by logging into your Linux 2.0.*x* or 2.2.*x* server as `root`, the superuser. Then follow the simple steps below. If you're going to use masquerading on a regular basis, you should integrate these commands into a system startup script in `/etc/init.d`, but the first time through it might be more educational to type them directly at a shell prompt.

1. Modern Linux kernels disable packet forwarding by default. This means that incoming IP packets will never be passed on to other computers. However, for masquerading we need to turn on packet forwarding, using the following command:

   ```
   echo 1 > /proc/sys/net/ipv4/ip_forward
   ```

2. If you use the most common type of PPP dialup account, where you get a new IP address every time you dial, you should turn on kernel support for dynamic IP address hacking, too:

   ```
   echo 1 > /proc/sys/net/ipv4/ip_dynaddr
   ```

3. (Optional.) By default, masqueraded connections that are idle for 15 minutes or longer "time out" and expire. This timeout duration can be adjusted; e.g., the following command sets the expiration timer to 2 hours (7200 seconds):

   (2.0) `ipfwadm -M -s 7200 10 160`

   (2.2) `/sbin/ipchains -M -S 7200 10 160`

4. (Optional.) Some IP-based protocols need extra kernel support for successful masquerading. You can load modules to support these protocols. For instance, to load the FTP masquerading module:

   ```
   modprobe ip_masq_ftp
   ```

   Besides FTP, masquerading modules exist for CuSeeMe, IRC, Quake, RealAudio, and VDOLive, among others. Note that most protocols do not require specific masquerading support.

5. Set default forwarding policy to "deny." This is very important; if omitted, then you allow anyone on the Internet to tunnel through your network, making it appear as if their network activity were originating from your own network.

   (2.0) `/sbin/ipfwadm -F -p deny`

   (2.2) `/sbin/ipchains -P forward DENY`

6. Turn on masquerading for your local network:

   (2.0)
   ```
   /sbin/ipfwadm -F -a m
       -S 192.168.0.0/255.255.255.0
       -D 0.0.0.0/0.0.0.0
   ```

   (2.2)
   ```
   /sbin/ipchains -A forward
       -s 192.168.0.0/255.255.255.0
       -j MASQ
   ```

   In the commands above you should replace `192.168.0.0` and `255.255.255.0` by the actual network address and netmask of your local network. If you're using my recommended setup, no changes are necessary.

That's it! You should now have a functioning IP masquerading setup. If not, refer to the following section.

3.1 Troubleshooting

This is where you should contact your geek. But if you want to try to debug it on your own, consider the following points:

Does the kernel support masquerading? A lot of distributions now come out-of-the-box with kernel support for masquerading. But yours might not. If you suspect that your kernel lacks IP masquerading support, refer to the IP Masquerading HOWTO for information on what options need to be enabled when recompiling the kernel.

Is the network card set up correctly? Can the server can `ping` other hosts in the network and that they can `ping` the server? If not, you've got a problem.

Are you online? Nothing in these instructions will make your server automatically auto-dial when a web page is requested. (That's explained later.)

Did you do the ritual sacrifice? Arachnae, goddess of networks, appreciates the sacrifice of hardware, the newer the better.

4 Client setup

Once you've set up the server, it's time to set up the clients. This is pretty easy. I'll cover how to set up Linux clients and Windows 9*x* clients. For more exotic systems, follow your nose, or refer to the IP Masquerading HOWTO.

4.1 Linux

Your distribution probably has some GUI method[6] for setting up networking. These instructions assume that you're not using that method. As a result, the following should work with just about any Linux kernel. It probably works, with little modification, under many other Unix-like kernels, too. Again, you'll want to modify your startup scripts to do the following automatically.

1. Set the machine's IP address, network address, and netmask. For instance:

   ```
   ifconfig eth0 192.168.0.2
            netmask 255.255.255.0
   ```

2. Tell the machine how to get to the local network. (Some newer kernels may set this up automatically.)

   ```
   route add -net 192.168.0.0 eth0
   ```

3. Instruct the machine that everything not on the local network should go through the server (it already knows how to get packets to the server, since it's on the local network; if it weren't, you'd have to tell it how), using it as a gateway:

   ```
   route add default gw 192.168.0.1
   ```

4. If you don't already have DNS set up, then create `/etc/resolv.conf` with the content shown below. Replace the values shown by your own domain(s) and local DNS server(s):

   ```
   search msu.edu debian.org gnu.org
   nameserver 35.8.2.41 35.8.2.42
   ```

[6] "It's GUI-riffic!"

4.1.1 Troubleshooting

If it doesn't work, you're doing something wrong. Duh. Your geek will know what's wrong. If he's on vacation, think about the following:

Can you see the server? Try using `ping` to test your connectivity to the server.

Is ICMP masqueraded? ICMP, the protocol used by `ping`, is only masqueraded by newer kernels. So if you're trying to ping the Internet through the server, it might not work. Try something else, like a web browser.

`route` is not idempotent. If you run `route` a lot, your routes will accumulate. You can view the routing table with `route` or `route -n`. To clear out the routing table, use `route del`. For more information, see route(8).

4.2 Windows 9*x*

Open "Control Panel" under "Settings" on the Start menu. Inside Control Panel, double-click on "Network." Within the main list box in the dialogue box that appears, find "TCP/IP" associated with your Ethernet card and double-click on it. Then make the following selections from the listed tabs:

IP Address Specify the client's IP address and netmask. For instance, you might use `192.168.0.2` and `255.255.255.0`.

Gateway Add a single entry, the address of your server. If you're following my recommendations, this is `192.168.0.1`.

DNS Configuration Select the "Enable DNS" setting. Enter a host name in "Host"; the name itself is not particularly important. Enter your preferred domain name within "Domain." Put your DNS servers in "DNS Server Search Order"; for instance, `35.8.2.41` and `35.8.2.42`. Enter any additional domains within "Domain Suffix Search Order."

Other Set as desired

You'll probably need to reboot after making these settings.

4.2.1 Troubleshooting

What do I look like, a Windows expert? "Fiddle with it until it starts working" is my advice.

There's probably no point in talking to your geek about it. He doesn't know, either.

5 Advanced server setup

If you've gone through what's above, you now have a working IP masquerading setup. Congratulations. But there's lots more that you can set up, with some extra work. You might be interested in setting up one or more of the following:

- Automatic dialing whenever Internet sites are accessed.
- Web and FTP caching to accelerate surfing.
- A DHCP server to simplify client setup.
- A nameserver to speed DNS lookups and allow naming client computers.
- An NTP client/server to synchronize machines' clocks.

The following sections briefly cover each of these possibilities.

5.1 Autodialing

Most of the time, what you really want to happen is for the modem to dial when you're using the Internet, then hang up when you're done. This is pretty easy to do using `diald`, and that's what I recommend that you use. I'll assume that you can set up `diald` on your own; it's not that hard.

But sometimes that's not what you want to do. For instance, you might want to keep the connection up permanently for a while, or you might want to prevent the modem from dialing entirely if you're using the line for other purposes. To conveniently allow for that, you can set up a helpful CGI script on your webserver.

A sample CGI script for this purpose is shown in Figure 2 on page 6. You should install this script into your web server directory (e.g., `/var/www`) and make it available only to machines on the local network.

In addition, you need to add a line to `diald`'s configuration file, of this form:

```
fifo /var/run/diald.fifo
```

The script has to be setuid to a user empowered to write to the FIFO mentioned above. Sometimes this has to be `root`, but you might be lucky enough that `dialout` or similar will work, too, or you may be able to adjust the ownership or permissions on the FIFO created by `diald`.

5.2 Web and FTP caching

Web browsers such as Mozilla keep a cache of sites recently visited by particular users from particular workstations in order to speed up web browsing. A web cache on a server keeps a cache of sites recently visited by anyone who uses the cache. This can significantly speed up web browsing over a limited-bandwidth connection.

To implement web and FTP caching, you just have to install and configure Squid on your server. This is a pretty easy process, and there's not much else to say about it.

I don't recommend using Squid on a machine with less than 32 MB RAM, and more RAM is better.

5.3 DHCP

I'm a programmer, therefore I'm lazy. I don't want to spend my time assigning and maintaining IP addresses for 18 different machines. Especially not for a laptop that often moves between networks. You probably don't want to, either. This is why you'd want to use DHCP for automatically assigning IP addresses. With Debian GNU/Linux, you'll want the `dhcp` package for this purpose; other distributions may have this DHCP server under a different name.

It's actually pretty easy to set up DHCP, but figuring out how to do it from the documentation can be gross. So here's a sample configuration file for our sample network setup:

```
server-identifier server.quux.org;
subnet 192.168.0.0 netmask 255.255.255.0 {
    range 192.168.0.128 192.168.0.254;
    option domain-name-servers 35.8.2.41;
    option domain-name "quux.org";
    option routers 192.168.0.1;
    option subnet-mask 255.255.255.0;
    option broadcast-address 192.168.0.255;
    default-lease-time 3600;
    max-lease-time 86400;
}
```

```
#! /usr/bin/perl

use CGI;
$q = new CGI;

if ($q->param ('action') eq 'up') {
    fifo_cmd ('unblock');
    fifo_cmd ('up');
    fifo_cmd ($q->param ('keep') ? 'force' : 'unforce');
} elsif ($q->param ('action') eq 'down') {
    fifo_cmd ('unforce');
    fifo_cmd ('down');
    fifo_cmd ($q->param ('keep') ? 'block' : 'unblock');
}

print $q->header(-expires=>'now');
print $q->start_html(-title=>'Modem Control');
print "<h1 align=center>Modem Control</h1>\n";
print $q->start_form ();
print "<table>\n";
print "<p><tr><td>action</td><td nowrap>bring the connection\n";
print $q->popup_menu ('action', ['up','down'], 'up');
print "</td><td>To dial the modem, select <strong>up</strong>, or\n";
print "to disconnect, select <strong>down</strong>.</td></tr><p>\n";
print "<p><tr><td>permanence</td><td nowrap>\n";
print $q->checkbox(-name=>'keep', -label=>'Keep it that way.');
print "</td><td>If you don't check this box, then the normal\n";
print "semantics will be applied to the connection. That is,\n";
print "if you selected <strong>down</strong> above, the modem will dial when\n";
print "there is Internet activity; if you selected <strong>up</strong>,\n";
print "the modem will disconnect after idle time.\n";
print "Checking this box will disable that behavior: the connection\n";
print "will stay up or down, as appropriate, until you come back to\n";
print "this page and resubmit the form.";
print "</td></tr><p>\n";
print "</table>\n";
print $q->submit ('Submit');
print "</p>\n";
print $q->end_form (), "\n";
print $q->end_html, "\n";

sub fifo_cmd {
    my ($cmd) = @_;
    open FIFO, ">/var/run/diald.fifo";
    print FIFO "$cmd\n";
    close FIFO;
}
```

Figure 2: CGI script in Perl to allow for manual control of `diald` autodialer. Also available from `http://www.msu.edu/user/pfaffben/masq/diald.cgi`.

By default, the DHCP server will hand out IP addresses on a first-come, first-serve basis within the range specified in the `range` directive above. You can also specify that particular machines should get fixed IP addresses, like this (be sure not to use an IP address reserved for dynamic assignment above):

```
host gameOSbox {
  hardware ethernet 08:00:07:26:c0:a5;
  fixed-address 192.168.0.5;
}
```

You can obtain the Ethernet address to specify on `hardware ethernet` from the output of `ifconfig` on the machine in question.

5.4 Nameserver

Up until now, you've had to refer to all of your client computers by their IP addresses. If you have more than just a few clients, you'll want to give them names instead. You can do this by setting up the BIND DNS server ("nameserver"), on your server machine. This also allows the server to cache DNS entries for the entire network, speeding up network operations.

Setting up BIND separates the lusers from the BOFHs. It is not easy. Fortunately, for your network, you won't need more than a simple setup, which as usual I've sugar-coated. My suggested BIND configuration file (typically `/etc/named.conf` or `/etc/bind/named.conf`) is this:

```
options {
    directory "/var/named";
    forwarders {
        35.8.2.41;
        35.8.2.42;
    };
    query-source address * port 53;
};

zone "." {
    type hint;
    file "named.root";
};

zone "localhost" {
    type master;
    file "named.local";
};

zone "127.in-addr.arpa" {
    type master;
    file "named.rev-local";
};

zone "quux.org" {
    type master;
    file "named.domain";
};

zone "0.168.192.in-addr.arpa" {
    type master;
    file "named.rev-domain";
};
```

You'll want to change the list of `forwarders` to show your ISP's nameservers, or you can leave them out entirely (but this will reduce performance). Note, in the last declaration of the file, that the network address for your local network `quux.org` is given in *reverse* order.

If you only want to run BIND for its DNS caching abilities, you can omit the last two `zone` entries above. You're done, at this point. Start BIND and get outta here.

Otherwise, you're interested in serving out your own domain. First choose a domain name. This doesn't have to be the name of a real domain and in fact shouldn't be. (If it is then you won't be able to access the corresponding real domain at all, by name at least.)

Start by replacing `quux.org` by this domain name above. Then you have to create two files. The first of these is `/var/named/named.domain`, and it should look something like this:

```
@ IN SOA ns.quux.org. root.quux.org. (
        1               ; Serial
        10800           ; Refresh
        1800            ; Retry
        3600000         ; Expire
        259200 )        ; Minimum
        IN      NS      server.quux.org.

server  IN      A       192.168.0.1
www     IN      CNAME   server
ftp     IN      CNAME   server
ns      IN      CNAME   server
proxy   IN      CNAME   server
client1 IN      A       192.168.0.2
client2 IN      A       192.168.0.3
client3 IN      A       192.168.0.4
```

The first line introduces the "SOA" or "start of authority" record that gives the name of the nameserver and the email address of the server administrator (`root@quux.org`) with '`@`' replaced by '`.`'. The second line is the file's "serial number," which should be incremented whenever its contents are changed.

The body of the file lists associations between hostnames and IP addresses using "A" records. There should be only one A record per host. It can also include hostname synonyms. There may be any number of these "CNAME" or "canonical name" records for a given host.

If you are using DHCP then you might wish to give unique names to all possible DHCP-assigned IP addresses. You should probably use a script to do this, unless you like typing in lots of repetitive data.

The second file you need to create is named `/var/named/named.rev-domain`. It should look like this:

```
@ IN SOA ns.quux.org. root.quux.org. (
        1               ; Serial
        10800           ; Refresh
        1800            ; Retry
        3600000         ; Expire
        259200 )        ; Minimum
        IN      NS      server.quux.org.

1       IN      PTR     server.quux.org.
2       IN      PTR     client1.quux.org.
3       IN      PTR     client2.quux.org.
4       IN      PTR     client3.quux.org.
```

This file is the inverse of the first one. The first part is identical in form and content to that in the first file. The remainder of the file specifies how IP addresses can be translated back into hostnames, with "PTR" records that "point" from IP addresses to hostnames. Every A record should have a corresponding PTR record, and vice versa.

There's a number of other possible DNS record types, but the above should be plenty for your masquerading network. For more information on setting up DNS and BIND, refer to the BIND documentation.

Once you have BIND set up, you can change the list of DNS servers on the server and the clients to point to your server. Notice that if you're using DHCP for your clients, all you have to do to make this change is to update your DHCP server configuration and either tell the clients to refresh from it or just wait for them to time out. For things like this, DHCP can be a real time- and headache-saver.

5.5 NTP Server

Sometimes it can be important to keep the clocks on all your machines synchronized. The easiest way to do this is to install `ntpd`, an NTP (Network Time Protocol) client and server.

The traditional way to do this is to have the NTP daemon running all the time. Unfortunately this doesn't work out too well for systems that aren't permanently connected to the Internet like yours.

An alternative is to add a call to `ntpdate` to your PPP startup script. The `ntpdate` program just retrieves, once, the current date and time from a remote server and sets the system clock to correspond. Of course, you still want to run the NTP server so that the other machines on the network can read the date from your server.

Local NTP time servers include MSU's nameservers `35.8.2.41` and `35.8.2.42`, also known as `serv1.cl.msu.edu` and `serv2.cl.msu.edu`.

6 Conclusion

Hypothetically, now you know how to set up a masqueraded network under Linux. Go to it!