djbdns for Dummies

by Gre7g Luterman

i. Table of contents

ii. Table of figures

iii.Table of tables

1. Introduction

First off, you're not dumb. You're administering your own Linux web server, and real dummies don't do that. Oh sure, Uncle Paul can install software on his Windows™ machine and Aunt Jenny can FTP her quilting web pages up to GeoCities™, but can they compile and install a .tgz? I think not. Even their little egghead son, Clyde, doesn't really understand ports and IP addresses.

But you do.

So why is it that you can't figure out djbdns? That's because all the documentation you can find for djbdns assumes you already understand BIND.

Sure, I want to shake my fist at Professor Bernstein and scream, "Why can't you explain how to do this?" But on further reflection, I think he's done quite enough just by creating the program itself.

That's why I decided to write this book[1]. It is my hopes that Joe-SysAdmin-who-really-doesn't-want-to-learn-everything-there-is-to-know-about-DNS,-but-needs-to-get-it-installed-and-working can use this to jump over the numerous hurdles laid out before him.

Note: This book contains a lot of my own personal opinions. It is not gospel. Your mileage may vary.

Further note: I am not and have never claimed to be an expert on djbdns or DNS in general. I've just recently had to learn it and make the mistakes you are probably making.

1.1. Who is this book for?

If you're still reading at this point, then this book is probably for you. Only masochists and übergeeks would want to learn djbdns for the sheer fun of it.

[1] Is it too short to call a book? I don't care. I'm not going to call it a "document". I'm going to call it a book, even if it ends up being only 10 pages long. How do ya' like them apples? No one is making you read this. If you want my help, you're gonna' call it a book too. Say it with me: B-O-O-K. Good.

DNS is cool in only the geekiest sort of ways, but it's really crucial to running an Internet-visible network.

This book is aimed at novice users who know their way around Linux boxes, but don't have a solid understanding of how DNS works. This book will be helpful for those users who understand DNS concepts, but have never configured a DNS system. It is for users who have heard how good djbdns is and want to jump straight into it instead of learning BIND and then learning how djbdns differs from BIND (what's the point, right?).

This book is an okay place to begin learning about DNS, but is not intended to be a reference for an advanced user, nor will it teach anything to an experienced djbdns user.

1.2. Why host my own DNS?

I can think of three good reasons why I host my own DNS. There are probably other good reasons, but these are mine.

1. I'm a control freak.

   I like knowing that I can buy a new domain name and configure my web server to serve the pages before internic has even distributed the new domain name.

   I hate having to e-mail a DNS provider to add a domain name to his registry, and I really hate waiting for them to get around to it.

2. I'm a tightwad.

   I hate spending lots of money on operating systems, web servers, databases, programming languages, etc. That's one of the coolest things about Linux; all the good stuff is free.

   I paid for the server, firewall, hub, domain names, and monthly DSL bill, so why should I pay for any extra service that I can do myself… for free?

3. I have a firewall with NAT.

   An external firewall is a great thing and I recommend one to everyone that asks. I love having the added security of local IP addresses that are not world-visible.

   But that means that the IP address I use to surf my websites from behind the firewall is different than the IP address used by computers out on the Internet. I could configure every machine on my LAN with a list of the locally hosted domain names, but that's a real pain in the butt. Wasting time is like wasting money. And having to update every machine on your LAN whenever you make a DNS change is wasting time in a big way.

1.3. Why use djbdns?

I use djbdns because BIND *evolved*. It was not written with security in mind.

Professor Bernstein explains this in great detail at http://cr.yp.to/djbdns/blurb/security.html, but I can sum it up pretty easily:

- BIND was written to perform a task and security was an afterthought. djbdns was written under tight control (it is free, but not GPL) with security as the focus.
- BIND is now (as of this writing) at version 9.2.0, mostly because people keep finding and exploiting security holes in it. I know this for a fact as one of my web servers got hacked this way.
- Professor Bernstein has offered a $500 cash reward for someone who finds a legitimate security hole in djbdns. I don't think anyone's ever claimed it and I don't think anyone will. If the ISC made the same offer on BIND, they'd go broke.

2. What does DNS do?

I know you already know the answer to this, but it would be bad form not to state it explicitly.

> Computers think in numbers and people think in names. DNS is the process of converting names into numbers, and vice-versa.

By using DNS:

- We humans can remember simple domain names (such as www.foo.com) and computers can be programmed to communicate with each other with IP addresses (such as 213.45.122.9).
- It becomes possible to change a server's location in the net without having to teach all the users a new address.
- We can balance the load of large, busy networks by creating multiple addresses that "mirror" the same domain name.

3. How does DNS work?

An explanation of how to configure DNS would be incomplete without at a lesson on how DNS works. I'll try to keep this short as my focus here is getting you up and running quickly.

3.1. How did DNS work originally?

Back in the early days of the Internet, back when there were only a handful of servers, every computer on the net had a flat text file which held a list of all the servers and domain names. It was a very simple solution for a relatively simple problem.

Flash forward to the present, and the bajillion different domain names that exist today. Remember not only to include names like microsoft.com, but also www.microsoft.com, support.microsoft.com, search.microsoft.com, developers.microsoft.com, and the dozens of others your typical web company might have.

Imagine a flat text file 3.2 godzillabytes in length to hold all those names. How long do you think it would take to scan that file every time you wanted to surf a page? How long would it take to distribute this *constantly* updated file to each of the 9.4 gajillion users on the net today?

Needless to say, the solution had to be revised. The database of domain names had to be distributed across the Internet, so that the servers only had to be responsible for answering questions about their own part of the 'net.

4. How does DNS work today?

The easiest way to picture DNS today is with an analogy.

4.1. The largest ball of twine

Suppose you want to see the world's largest ball of twine. You break out your handy copy of *The Guinness Book of World Records* and you look up "twine, largest ball of". This book, the authority you trust, says that the largest ball of twine can be found in Cawker City, Kansas (of course).

It's a beautiful spring day and you don't have anywhere to be, so you jump in the car and start driving. Pretty soon, you come across an authority you trust (a gas station attendant). You ask him "How do I get to the world's largest ball of twine in Cawker City, Kansas?"

The gas station attendant has no idea where the largest ball of twine is, but he's not clueless. He has been entrusted with a wealth of directional knowledge and he gladly shares it with you. "Kansas is 'dat boring looking state there in the middle of yer' map," he explains. "Topeka is the capital of Kansas, ya' know? They'll know how to get you there."

So you grab your snacks and piping hot coffee and drive across the country to Topeka, Kansas. You drive right up to the capitol, park on the steps, and strut up to the information desk, saying, "How do I get to the world's largest ball of twine in Cawker City, Kansas?"

The information booth gal looks blankly at you. She doesn't know where the largest ball of twine is, but she's not clueless. She has also been entrusted with a wealth of knowledge that she gladly shares it with you. "Cawker City is that teeny-tiny, itsy-bitsy dot on your map," she explains. "The people at the Cawker City town hall will know how to get there."

So you drive to the Cawker City town hall, either blissfully unaware of the billboards begging you to see the largest ball of twine, or perhaps they were all knocked down by Dorothy's house or something. Regardless, you drive up to the Cawker City, town hall and ask the authority where you can find the largest ball of twine, located in Cawker City, Kansas.

He leans on his broom and stares blankly at you. You're getting used to that look now, but this guy does know the answer. He lifts an arm and points at the largest ball of twine you've ever seen in your life.

4.2. http://www.balloftwine.com

Now, it's time to draw the analogy back to DNS! This time we're going to look for http://www.balloftwine.com[2].

When you began your trip, you didn't know where to go (the IP address for www.balloftwine.com was not cached) so you started by consulting an authority (*The Guinness Book of World Records*). In the computer version of this tale, you would consult a root server and ask them.

There's about a dozen root servers in the world and the IP addresses of each are programmed into every resolver (the program that searches for an IP address based on domain name). The resolver picks one at random (to keep the load on each balanced) and posts a query. We'll choose the root server at 192.36.148.17 for this example:

```
$ dig +norec @192.36.148.17 www.balloftwine.com
```

The book didn't tell you the exact answer (the street location), but it did tell you something of use. Likewise, the root server didn't have the IP address, but it did give you the addresses of other servers that will have more information. Like with the root servers, we can pick any from the list and continue by asking them.

```
;; AUTHORITY SECTION:
com.                 2D IN NS    H.GTLD-SERVERS.NET.

;; ADDITIONAL SECTION:
H.GTLD-SERVERS.NET.  2D IN A     192.54.112.30
```

The root server recommended we try h.gtld-servers.net at an IP address of 192.54.112.30. h.gtld-servers.net is apparently an authority on domains ending in "com".

The gas station attendant pointed you to Kansas. On the net, the resolver would now consult h.gtld-servers.net.

```
$ dig +norec @192.54.112.30 www.balloftwine.com
```

[2] This website is actually about a largest ball of twine in Minnesota. I don't know what to make of that. Perhaps I should have actually checked with Guinness instead of just looking on the web. I have to admit that the Minnesota ball *looks* bigger than the Kansas ball. Perhaps the guy standing next to the Minnesota ball is a midget or something. I'm sure it's a very bitter rivalry in the world of twine balling, and I can see any state stooping to such a low level as to dress their twine ball up, but I've already done more looking into this than I've planned to.

Again, this query returns multiple servers and we can choose any:

```
;; AUTHORITY SECTION:
balloftwine.com.      2D IN NS    DNS39.REGISTER.com.

;; ADDITIONAL SECTION:
DNS39.REGISTER.com.   2D IN A     216.21.234.90
```

Asking the gal in Topeka would be like asking the dns39.register.com server.

```
$ dig +norec @216.21.234.90 www.balloftwine.com
```

This time, the dig command returns not only authorities and their IP addresses, it also returns an answer:

```
;; ANSWER SECTION:
www.balloftwine.com.  1D IN CNAME balloftwine.com.
balloftwine.com.      1D IN A     209.67.50.203
```

This query returned two answers. First, it told us that www.balloftwine.com was a CNAME of balloftwine.com. A CNAME is an alias, so www.balloftwine.com is the same as balloftwine.com.

Secondly, since we were sure to ask where balloftwine.com was, dns39.register.com goes ahead and volunteers the address for balloftwine.com (209.67.50.203).

In case you're curious, the "1D IN" means that the data may be cached for one day (1D) and that it is an Internet (IN) address. Dig can fetch a couple other types of records, but they've all been obsoleted, so don't worry about them.

It appears my analogy ran a little short here, so I guess asking the janitor in Cawker City would be like doing a web page fetch from 209.67.50.203. Anyhow, I'm sure you get the idea. The resolver starts with the root servers and keeps asking questions until it finds the address we were looking for.

We call this sort of resolver a recursive resolver because it will ask each server it needs to, to get a final answer. Some resolvers are not recursive and will only return the next "hop" in the search. You might have noticed the "+norec" in our dig commands. That instructed it not to recurse, so that we could see each step of the process.

The final name server consulted in the above example was dns39.register.com. Our goal in this book will be to set up our own name servers, instead of using a pay service like the one provided by register.com.

In this book, you will learn to set up name servers ns.yourdomain.com and ns2.yourdomain.com; and point them to yourdomain.com. We'll also point the .com server to your name servers and then people will be able to surf www.yourdomain.com or ftp to ftp.yourdomain.com or mail you at mail.yourdomain.com.

4.3. Caching

The world's largest earlobe is Cawker City's newest attraction and drawing tourists like you could only imagine. Now that you're here, you decide to check it out.

There's really no point in driving back to Topeka to find out where Cawker City is; you already know that. Similarly, if we want to find www.earlobe.com, there is no reason to bother the root servers. After all, we already know that h.gtld-servers.net is an authority on domains ending in "com".

4.4. Time to live (TTL)

Cached information should not be cached forever. Websites move around from time to time. So all information received in a DNS search has a TTL.

After the information has been cached for however long indicated by the name server, it must be discarded. Future web searches will have to repeat the query if they need the information again.

4.5. What's a reverse DNS lookup?

A reverse lookup is just what it sounds like; you take the IP address of a machine and find out the machine's name. The process is very similar, not terribly interesting, not terribly important, and discussed much later (see Section 10).

If you host multiple domains like I do, you probably have more domain names than IP addresses. Since each reverse DNS returns only one domain name, not all domains I host will be represented. That's quite okay.

Why bother? Well some lame mail servers won't accept mail addressed from ed@balloftwine.com if the sender's IP address is strange.oddities.com.

This is very lame indeed, but just how things are. If you find too much of your mail being bounced, you'll need to send mail from the address represented in the reverse lookup.

Also, the reverse lookup can be useful when complaining about spam. If you have the sender's IP address, you can look up a server name and post some complaints.

5. How do I get a domain name?

I'm *really* going to gloss over this one because even Aunt Jenny can figure out how to register a domain name. It's really quite easy. Pick a registrar, fill out a form, and give them money.

You will want to pick a registrar that won't go out of business, one that won't charge too much, and one that makes it easy to update your records. You can find a long list of registrars at http://www.internic.net.

NetworkSolutions.com is probably the most well known of the registrars, but I despise them. They charge too much and they make it too hard to change your registration.

I prefer Dotster.com, but you're welcome to go anywhere.

> **Tip:** When you register a domain name, you will need to give them an e-mail address. If you're registering bar.com, then giving them foo@bar.com would be a bad idea. How will you get any mail they send you during the registration process?
>
> Once your registration is complete and your server is running, I would still recommend against changing your registered e-mail to foo@bar.com. This is just asking for trouble. If you get hacked or your server goes down, this could complicate getting things fixed.

The only thing remotely tricky about getting a domain name is when you have to fill out the name servers. If you were paying someone else to host your name service, you would just put in the name servers they use. Obviously you don't want to do that, or you wouldn't be reading this book.

Anyhow, your registrar isn't liable to accept ns.bar.com and ns2.bar.com if you haven't registered bar.com yet. And how can you register bar.com without entering name servers? Yup. That's a problem all right. A friend of mine calls this "the chicken and the egg," and neither of us have a perfect contingency plan to deal with it.

The two obvious solutions are:

1. Get someone else to host your name service long enough to get everything up and running.

2. Lie about your name servers.

Clearly, "1" is the correct solution. I'm not saying "2" won't work, it just has potential badness written all over it.

I suppose you could put in known good name servers like Microsoft's dns1.cp.msft.net, and dns1.tk.msft.net as placeholders, and then change them to ns.bar.com and ns2.bar.com later, but I can't say for a fact whether you will be able to get ns.bar.com and ns2.bar.com to register if the Microsoft's name servers won't correctly identify you.

You're on your own here.

6. Installing djbdns

Installing djbdns is not so bad. It installs pretty much just like the instructions tell you.

1. Install daemontools to monitor your DNS programs and restart them if they die for some reason.

   (http://cr.yp.to/daemontools/install.html)

2. Skip the afxr stuff. Anything about zone transfers is for dealing with sharing configurations between servers. Professor Bernstein recommends you use rsync to do this instead, if you need to share configurations.

3. Install djbdns. *Just* install it. Don't skip ahead on the following web page to where they're talking about configuring. Heed my words! Before you go configuring anything, we need to figure out what sort of configuration you require!

   (http://cr.yp.to/djbdns/install.html)

7. Mapping your LAN/WAN

DNS provides information about how your network is set up. How are you going to tell the DNS program how your network is set up if you don't know? You can't.

Tip: Don't guess how your network is set up. I got myself in lots of trouble here by trying that. Take the time to build a good, accurate map of your network. You don't have to get all fancy, but you will save yourself a lot of problems down the road if you create and maintain a good map.

7.1. Simple networks

Even with a simple network as shown in Figure 1, it's worth the minute it will take to draw a picture and write down:

- Your machine's name (foo)
- Your static IP addresses (12.165.54.198 & 12.165.54.199)
- Your name servers (ns.bar.com & ns2.bar.com)
- The websites you host (bar.com)
- Any other domain names you serve, but do not host, and the IP address they are served at (sneakers.org, 54.45.66.102)

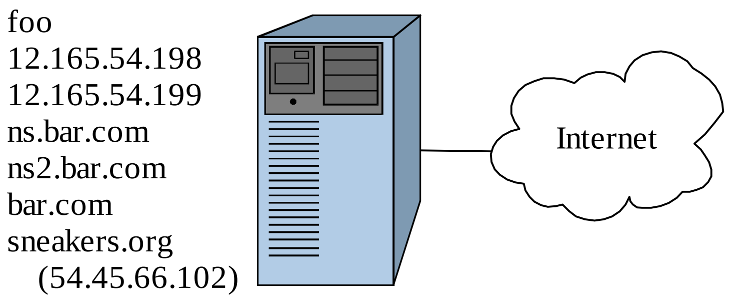

Figure 1 - A Simple Network

7.1.1. Machine name

Every machine on your LAN/WAN should have a simple, unique name. Even if the computer's only reason to exist is to serve up web pages for bar.com, don't try calling it "bar.com". I've made that mistake. It will let you do it, and then you'll waste a hundred hours trying to figure out why it is acting weird. Give it its *own* name.

Also, don't try naming it "www". That's a bad idea, and it invites problems. It makes things confusing. Are you talking about the computer "www.bar.com" or the domain name "www.bar.com"? Bleah! Naming a web server "www" is begging someone else to set up a secondary web server and call it "www" also.

Name your computers larry, moe, and curly. Name them after states or elder gods. You can name them after the employees who use them, but that leaves a black hole for the servers (since they're not really used by a single employee) and it causes problems when two people have the same name. Animals are always good for computer names. There's lots of animals in this world and it lets people express themselves in nice, safe ways.

If you maintain a larger network of multiple domains, you could technically re-use computer names by placing the machines under different domains. cow.big.animals.com is a different machine than cow.farm.animals.com. However, I recommend you do *not* do this. It will work, but it gives you more opportunities to screw up.

I know of two different places to place your computer name on a Linux machine. There may be some others, but these are the ones I know of: /etc/defaults/rc.conf and /etc/hosts

Your /etc/defaults/rc.conf file should contain a line that looks like this:

```
hostname="<long name>"
```

You will need to replace the <long name> portion of this line with your computer's FQDN (fully qualified domain name). For our example in Figure 1, that would be "foo.bar.com".

Your entire /etc/hosts file should look something like this:

```
127.0.0.1              localhost
<internal IP address>  <name>
```

For our example, we would replace <internal IP address> with "12.165.54.198", and <name> with "foo".

7.1.2. Static IP addresses

If you're going to run a server, you really need static addresses. You can't go reconfiguring the entire web every time your dynamic IP changes.

Also note that you're going to need at least two addresses. When you register a domain name, they always want at least two name servers, and I don't think they'll let you enter the same IP address twice.

This doesn't mean that you'll need two computers. You can put many IP addresses on a single PC, but you will need at least two static IP's.

The two IP addresses in Figure 1 are real, external Internet addresses. If you're not going to use a firewall that gives you a private LAN (with internal addresses like 192.168.168.*), then make sure you run a software firewall inside your box. The 'net is just too dangerous a place to hang an unprotected server.

7.1.3. Name servers

For simplicity and recognizability, I always use ns and ns2 as my name server prefixes. If you own bar.com, then use ns.bar.com and ns2.bar.com. You can have more than two, but unless you are serving a huge amount of pages[3] there is really no need.

7.1.4. Websites

Keep a list of all the websites you host, even if that list is only a single site. Why? Because the list will grow. It always does. Just like buying a pickup truck causes one of your friends to move or washing your car causes it to rain, if you host a site, you will add on another.

You might as well use that bandwidth, right? And after all, it only takes a little bit of work to add on a second site after you jump through all the hoops to set up the first one.

7.1.5. Domain names

I'm guessing that you'll host the domain names of every site you host (why else would you be reading this book?), but it's not uncommon to also end up hosting names for people too. Think of how much you would have liked a friend to host your domain name back when you were fighting the "chicken and the egg" in Section 5!

7.2. More complex networks

If you have a more complex network, like the one shown in Figure 2, you need to do the same basic things. Draw a detailed picture and keep track of all the important "networky" things you can think of.

If you've got a complex enough network, consider dropping a couple bucks on SmartDraw or other drawing program. Print out the pages on several sheets of paper and tack them to your cube

[3] And if you *are* serving a huge amount of pages, then you should really be paying a professional to do this for you, and not waste your time with a Dummies book. Saving money is one thing, but don't skimp if you have a major site!

wall for easy reference. Yeah it's pretty geeky as wallpaper goes, but you'll be surprised how it reminds your friends in accounting that they don't get to do anything cool.

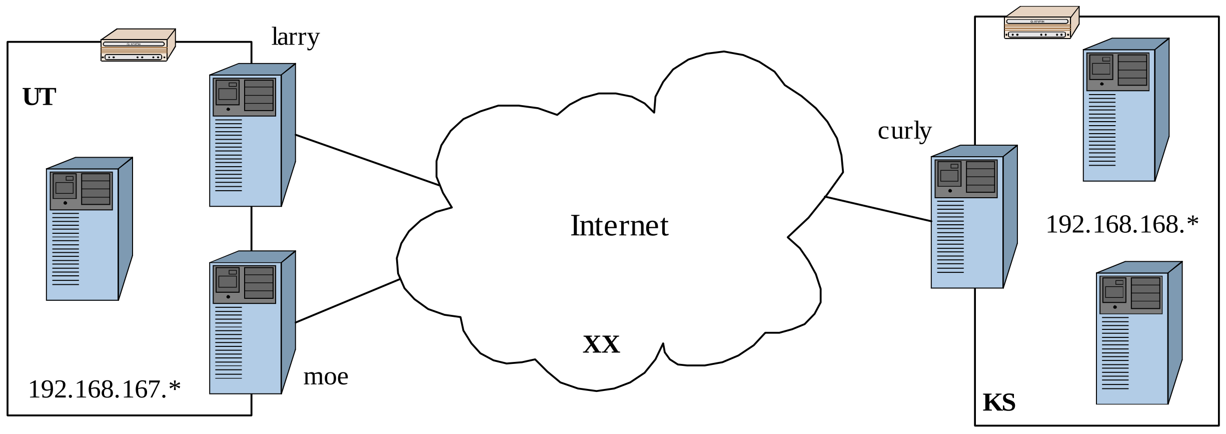

Figure 2 - More Complex Network

Note how I drew the firewalls not just as a unit, but as a border around the computers they protect. Servers that are externally visible (using a one-to-one NAT) also cross the firewall border since they exist in both regions.

You might want to skip drawing all the network connections as they can make your drawing a real mess.

Track all the information we discussed in Section 7.1 (I didn't show them because my drawing is small) *plus*:

- Routers and hubs (I personally wouldn't go so far as to track which computer is plugged into what port, but I would at least track where the device is physically located)
- Physical location of computers
- DHCP address ranges for all dynamic IP regions
- Internal and external IP's (I draw them on the appropriate sides of the firewall line so as to make it extra clear which is which)
- Which server services are on the servers (HTTP, FTP, etc.)

7.3. Cache

Now that you know what you've got in your network, you will need to place DNS caches (as discussed in Section 4.3). One cache can serve multiple computers, but the cache should be located "nearby" to the computers using it.

If your network looked like the one shown in Figure 2, you *could* make larry your only DNS cache. This would work, but it would mean that computers in the 192.168.168.* LAN would have to ask larry for their DNS information. That's an extra delay that could easily be removed by making both larry and curly DNS caches.

LAN's may have multiple caches and PCs may be configured to consult up to three servers for their DNS information. In other words, you may wish to make all three of the servers in Figure 2 (larry, moe, and curly) DNS caches. This will provide your users a little more robustness. The users on 192.168.167.* will still have access to a cache even if larry goes down.

However, if you have many servers on a LAN, I wouldn't bother, making every one of them a cache. A single backup cache should suffice unless your servers go down regularly.

7.3.1. Local cache

If your entire network looks like the one in Figure 1, and you have no intention of adding additional computers, then you will only need a local cache.

Install it as described at http://cr.yp.to/djbdns/faq/cache.html#config.

7.3.2. External cache

If your network is more sophisticated than the one in Figure 1, then you will want to configure external cache(s) and you will have no need of internal caches.

External caches must be assigned to a machine and an IP address. Furthermore, they can not be on the same IP address as a name server.

For simplicity and safety, I recommend you always create a new IP address for your external caches, rather than try to place them on a server's normal, IP address. This will make it easy to configure your firewalls so that the caches are not visible to the outside world (why provide a caching service to other users on the net, anyhow?).

Let's suppose you wanted to place caches in all three of the servers shown in Figure 2. Here's some typical values you might have:

Server	Ext. IP	Int. IP	Cache IP
larry	24.37.254.19	192.168.167.10	192.168.167.11
moe	24.37.254.20	192.168.167.20	192.168.167.21
curly	172.54.55.199	192.168.168.10	192.168.168.11

Table 1 - IP Addresses

Caches can be configured as described at http://cr.yp.to/djbdns/faq/cachex.html#config.

Do this for each machine that will host a cache.

Note: Since I can't see a good reason for your average server to have both an internal and an external cache, I named my cache service directory dnscache instead of the dnscachex as in the configuration instructions. Feel free to do the same if you like.

7.3.3. resolv.conf

Linux boxes will need to be configured to use the DNS cache. This is done with a file at /etc/resolv.conf.

Edit /etc/resolv.conf and enter the following:

```
nameserver <IP address>
```

/etc/resolv.conf is configured with a few simple commands, each on a line by themselves. If you have both a primary and secondary DNS cache, simply list a second nameserver line below the first. The computer will search the name servers in order.

If you installed a local cache instead of an external cache, use 127.0.0.1 as the name server's IP address.

8. Configuring the DNS

Now we're finally ready to begin configuring the DNS.

tinydns (the program that answers resolver queries) is configured with a file typically located at /service/tinydns/root/data. You can add entries to this file by calling programs like add-ns and add-host.

I prefer instead to do this by manually editing the file. This lets me add comments and group the entries in a more logical, readable way. Making this file readable is a very good thing for those of us SysAdmins who don't have DNS changes they need to make every day.

Typically, I add or change a DNS record no more often than once every few months, and so by the time I need to mess with the file again I've forgotten what all the different record types and commands mean. By keeping this file tidy, I can make changes simply, without digging deeply into the documentation to relearn how things work.

Begin configuring tinydns on each machine you plan to use as a name server. The instructions to do so are shown in http://cr.yp.to/djbdns/faq/tinydns.html#config, but stop after the part where the service starts running. In the following sections we'll start writing a generic configuration file that can be used on each name server.

8.1. Network regions

Once you've mapped your network, it should be easy to see how many regions you have. Typically, you will have one more region than you have firewalls.

If your network has only one region (like in Figure 1) you can totally skip this step.

In Figure 2, for example, there are three regions; one for addresses 192.168.167.*, one for the Internet, and one for addresses 192.168.168.*.

Pick some names for your regions. djbdns only allows one and two letter names, so I picked UT for the computers in Utah, KS for the computers in Kansas, and XX for the computers on the Internet. Yes, you need a region for the Internet, even if none of *your* computers are out there.

If you plan to have only one firewall for your entire network, then I recommend you use the standard "IN" and "EX" for your internal and external regions.

For the example shown in Figure 2, begin the /etc/tinydns/root/data file off with:

```
# Network regions
%UT:192.168.167
%KS:192.168.168
%XX
```

So far the file it pretty self-explanatory. The regions are tested in the order listed, so XX (not having any IP address range restriction) catches anything that isn't included in the other two regions.

For a simple internal/external arrangement, you could use:

```
# Network regions
%IN:192.168
%EX
```

That would put all IP's in 192.168.* in region IN and all other IP's in region EX.

8.2. Named computers

Now it's time to identify the IP addresses of every "visible" computer on your network. I stressed the word visible because not every computer needs to be identified, just those that need to be found by name. Typically, that means your servers, not your user machines.

Be sure to include the computers that are running web servers, name savers, and FTP servers. Identify every computer that is visible from the Internet.

These computers must be identified by their IP addresses, but which IP address do you use? The internal address or the external? The answer is both. You will want to identify larry's address (for example) not only to the local users on 192.168.167.*, but also to the internet users and users on 192.168.168.*.

8.2.1. Simple networks

If you have a simple network like the one shown in Figure 1, you would need only list that in your /services/tinydns/root/data file:

```
# Computers
=foo.bar.com:12.165.54.198:3600
=foo2.bar.com:12.165.54.199:3600
```

Computer foo had two IP addresses, so I listed foo as the primary and foo2 as the secondary. Feel free to be even more creative with your setup.

The only other part remaining is the mysterious "3600" appearing in each record. That is our TTL value in seconds. I've arbitrarily chosen an hour (3600 seconds) as how long the records should be cached before being discarded.

Another popular value for TTL is 86400 (one day). You are not restricted to using one of these two values, but I personally wouldn't use anything smaller than 3600 or larger than 86400. Making the value too small causes the DNS caches to be somewhat ineffective. Making the value too large results in long, sloppy changeover periods if you ever have to move a computer from one IP address to another (if you change ISP's, for instance).

8.2.2. More complex networks

Let's suppose we needed to configure the IP addresses shown in Table 1, for the setup in Figure 2. We would list them in the /services/tinydns/root/data file as such:

```
# Computers
=larry.yourdomain.com:192.168.167.10:3600::UT
=larry.yourdomain.com:24.37.254.19:3600::XX
=larry.yourdomain.com:24.37.254.19:3600::KS
=moe.yourdomain.com:192.168.167.20:3600::UT
=moe.yourdomain.com:24.37.254.20:3600::XX
=moe.yourdomain.com:24.37.254.20:3600::KS
=curly.yourdomain.com:172.54.55.199:3600::UT
=curly.yourdomain.com:172.54.55.199:3600::XX
=curly.yourdomain.com:192.168.168.10:3600::KS
```

Notice how every possible combination of computer and region is included. We listed internal addresses for local regions, and external addresses for users on the Internet and located in different regions.

Now if a user within the Utah region (192.168.167.*) wants to access larry, they'll be given the internal address of 192.168.167.10 instead of the (unreachable) external address of 24.37.254.19. Internet (XX) users will always be given external addresses (obviously) as internal addresses are not routable.

I've assumed that all three of the servers are in the same domain (yourdomain.com). That doesn't really need to be the case if you have multiple domains. If moe, for instance, was a web server for otherdomain.com, you could just as easily have listed those three records as:

```
=moe.otherdomain.com:192.168.167.20:3600::UT
=moe.otherdomain.com:24.37.254.20:3600::XX
=moe.otherdomain.com:24.37.254.20:3600::KS
```

8.3. External caches

If you configured any external caches, you will want to add them to your /services/tinydns/root/data file. I've cleverly named the caches from Table 1: cache1, cache2, and cache3.

```
# External caches
=cache1.yourdomain.com:192.168.167.11
=cache2.yourdomain.com:192.168.167.21
=cache3.yourdomain.com:192.168.168.11
```

Notice that we only listed internal IP addresses this time. We don't plan to make our caches visible to the outside world, so there is no point listing internal and external addresses.

8.4. Name servers

Lets suppose that larry and curly will be our name servers in Figure 2. We immediately have a problem because when we registered yourdomain.com, we told the registrar that our name servers would be ns.yourdomain.com and ns2.yourdomain.com.

We *could* have registered larry.yourdomain.com and curly.yourdomain.com, or we could have named larry ns and named curly ns2. I would recommend against either of these solutions. A better plan is to make ns.yourdomain.com an alias for larry.yourdomain.com and ns2.yourdomain.com an alias for curly.yourdomain.com.

We could do that by adding the following lines to /services/tinydns/root/data:

```
# Nameservers
+ns.yourdomain.com:192.168.167.10:3600::UT
+ns.yourdomain.com:24.37.254.19:3600::XX
+ns.yourdomain.com:24.37.254.19:3600::KS
+ns2.yourdomain.com:172.54.55.199:3600::UT
+ns2.yourdomain.com:172.54.55.199:3600::XX
+ns2.yourdomain.com:192.168.168.10:3600::KS
```

This should look almost exactly like the definitions of computers larry and curly.

For the network shown in Figure 1, your alias definitions would be:

```
# Nameservers
+ns.bar.com:12.165.54.198:3600
+ns2.bar.com:12.165.54.199:3600
```

Again, there is no need to list different regions in a one region network.

8.5. *Reverse DNS lookup*

I'm not quite ready to discuss how reverse DNS works, but this is where we put the reverse DNS definitions into our /services/tinydns/root/data file. So instead of going into lots of detail, I'm just going to give you some instructions and I recommend you follow them blindly, for now. I'll try to explain what they actually meant when we get to Section 10.

The first thing you have to do when you define your reverse DNS mappings is to make a list of all IP ranges used in your system, and convert them into a reverse DNS format. This looks very weird, but is quite easy. All you have to do is reverse the order of the octets (those numbers in an IP address, separated by periods), and tack on ".in-addr.arpa".

8.5.1. More complex networks

We used four different IP address ranges in Figure 2. Those ranges and the reverse DNS formats for those addresses are:

IP range	reverse DNS format
24.37.254.*	254.37.24.in-addr.arpa
172.54.55.199	199.55.54.172.in-addr.arpa
192.168.167.*	167.168.192.in-addr.arpa
192.168.168.*	168.168.192.in-addr.arpa

Table 2 - Reverse DNS format

Like I said, it looks weird, but it's simple enough to do.

You may notice that I did not merge 192.168.168.* and 192.168.167.* into one big range of 192.168.*. I'll try to explain why that is important later, in Section 10.

Now that we have our reverse DNS formatted ranges, we need to create entries in our /services/tinydns/root/data file. We will need one entry for each range, for each name server (i.e. four ranges x two name servers = eight definitions).

For our setup, we would add the following lines:

```
# Reverse DNS lookup
# Externals
.254.37.24.in-addr.arpa:24.37.254.19:a
.254.37.24.in-addr.arpa:172.54.55.199:b
.199.55.54.172.in-addr.arpa:24.37.254.19:a
.199.55.54.172.in-addr.arpa:172.54.55.199:b
# Internals
.167.168.192.in-addr.arpa:192.168.167.10:a
.167.168.192.in-addr.arpa:172.54.55.199:b
.168.168.192.in-addr.arpa:24.37.254.19:a
.168.168.192.in-addr.arpa:192.168.168.10:b
```

Yuck, right? Well, let's run over it real quickly and break it apart.

Each reverse DNS formatted range is listed twice, once for each of the two name servers (larry and curly). The "a" and "b" codes at the end of the line are used to distinguish between records for larry and curly (larry is "a" and curly is "b").

The middle part of each line is the name server's IP address. "a" records list the IP address for larry, and "b" records list the IP address for curly. If you were using three name servers, we would have "a", "b", and "c" records.

The first four records (just under the "Externals" comment) are the reverse DNS formatted external IP address ranges. Likewise, the name server IP addresses used are also the external ones for larry and curly.

The last four records (just under the "Internals" comment) are the reverse DNS formatted internal IP address ranges. As you would probably expect by now, two of the name server IP addresses used are the internal ones for larry and curly. So why do the other two records point to external addresses? To answer that question, you'll need to refer back to Figure 2 and Table 1.

Suppose moe asks about the domain at 192.168.167.10. We know the *answer* to the question is "larry.yourdomain.com", but tinydns must also supply the list of name servers that hold that information. Why isn't the answer good enough? I don't know. Anyhow, the name servers that can identify 192.168.167.10 are larry and curly.

We will want to identify larry with its internal address because we know the request was made from within the firewall (if it had been made from outside the firewall, the requester would have asked about larry's external address 24.37.254.19). But when we identify curly, we need to use curly's *external* address because the requester won't be able to talk to curly with curly's internal address. Since the request obvious came from within the UT region, we must use an external address for curly so any requests to curly get routed across the Internet.

Yes, this is odd, but it does make sense if you think hard enough about it. If you set up your list correctly, you should have N records that use internal addresses (where N is the number of name servers you run), and all the remaining records should refer to external addresses.

8.5.2. Simple networks

For our simple network shown in Figure 1, we only need the following:

```
# Reverse DNS lookup
.54.165.12.in-addr.arpa:12.165.54.198:a
.54.165.12.in-addr.arpa:12.165.54.199:b
```

The process is exactly the same, but there's only one range of IP addresses and two name servers.

8.6. Domain names

I was starting to think we'd never get to this section! The whole point of doing DNS is to provide domain name mapping and this is the section where we'll describe what domain names are hosted where. Are you pumped? I am, so let's begin.

8.6.1. More complex networks

First, I'd like to re-stress organization. If you keep your configuration file neat and tidy, it should make life easier in the long run. The way I do this is by organizing my sites by server, and then by sorting the remaining sites alphabetically.

For instance, let's say that we were going to host the following websites:

Domain	Server
earwax.org	larry
toejam.net	larry
otherdomain.com	moe
yourdomain.com	moe
iloveskunks.com	curly

Table 3 - Domains hosted

Notice how I organized the sites into groups and then sorted the sites by domain name. Admittedly, this isn't a very long list, but the list tends to grow.

For each domain, you will probably want to list three types of records:

1. NS records that identify the name servers for the domain

2. Alias records so that in addition to yourdomain.com, you also handle www.yourdomain.com, ftp.yourdomain.com, etc.

3. MX records that identify where to send your mail.

Here's what I would enter for these servers in our imaginary system:

```
#
# Sites hosted on: larry
#

# Site:          earwax.org
# Administrator: Jimmy Earwax (jim@earwax.org)
# Registrar:     dotster.com
.earwax.org::ns.yourdomain.com:3600
.earwax.org::ns2.yourdomain.com:3600
+*.earwax.org:192.168.167.10:3600::UT
+*.earwax.org:24.37.254.19:3600::XX
+*.earwax.org:24.37.254.19:3600::KS
@earwax.org:192.168.167.10:earwax.org::3600::UT
@earwax.org:24.37.254.19:earwax.org::3600::XX
@earwax.org:24.37.254.19:earwax.org::3600::KS

# Site:          toejam.net
# Administrator: Jimmy Earwax (jim@earwax.org)
# Registrar:     dotster.com
.toejam.net::ns.yourdomain.com:3600
.toejam.net::ns2.yourdomain.com:3600
+*.toejam.net:192.168.167.10:3600::UT
+*.toejam.net:24.37.254.19:3600::XX
+*.toejam.net:24.37.254.19:3600::KS
@toejam.net:192.168.167.10:toejam.net::3600::UT
@toejam.net:24.37.254.19:toejam.net::3600::XX
@toejam.net:24.37.254.19:toejam.net::3600::KS


#
# Sites hosted on: moe
#

# Site:          otherdomain.com
# Administrator: John Smith (jsmith7@hotmail.com)
# Registrar:     networksolutions.com
.otherdomain.com::ns.yourdomain.com:3600
.otherdomain.com::ns2.yourdomain.com:3600
+*.otherdomain.com:192.168.167.20:3600::UT
+*.otherdomain.com:24.37.254.20:3600::XX
+*.otherdomain.com:24.37.254.20:3600::KS
@otherdomain.com:192.168.167.20:otherdomain.com::3600::UT
@otherdomain.com:24.37.254.20:otherdomain.com::3600::XX
@otherdomain.com:24.37.254.20:otherdomain.com::3600::KS
```

```
# Site:          yourdomain.com
# Administrator: Jacob Marley (jmarley@yourdomain.com)
# Registrar:     networksolutions.com
.yourdomain.com::ns.yourdomain.com:3600
.yourdomain.com::ns2.yourdomain.com:3600
+*.yourdomain.com:192.168.167.20:3600::UT
+*.yourdomain.com:24.37.254.20:3600::XX
+*.yourdomain.com:24.37.254.20:3600::KS
@yourdomain.com:192.168.167.20:yourdomain.com::3600::UT
@yourdomain.com:24.37.254.20:yourdomain.com::3600::XX
@yourdomain.com:24.37.254.20:yourdomain.com::3600::KS

#
# Sites hosted on: curly
#

# Site:          iloveskunks.com
# Administrator: Pedro Gonzales (skunky@iloveskunks.com)
# Registrar:     register.com
.otherdomain.com::ns.yourdomain.com:3600
.otherdomain.com::ns2.yourdomain.com:3600
+*.otherdomain.com:172.54.55.199:3600::UT
+*.otherdomain.com:172.54.55.199:3600::XX
+*.otherdomain.com:192.168.168.10:3600::KS
@otherdomain.com:172.54.55.199:otherdomain.com::3600::UT
@otherdomain.com:172.54.55.199:otherdomain.com::3600::XX
@otherdomain.com:192.168.168.10:otherdomain.com::3600::KS
```

Whew! That was a lot of typing.

As you can see, the NS records (which begin with a ".") define the same name servers for every site. If configured correctly, each of these sites should be registered with the two name servers shown, ns.yourdomain.com and ns2.yourdomain.com.

The records beginning with a "+" are alias records, like the ones we used in the name servers section. The "*." in these records tell tinydns that any prefix added on to the domain name should be treated like the domain name itself. You will still have to configure Apache (or whatever web server you use) to respond to different prefixes, but these aliases will make sure the requests get there.

Finally, the MX records (beginning with "@") insure the mail ends up on the right server. If you are not running a mail server on every box, you will change these records to point to the correct mail server.

8.6.2. Simple networks

You can probably extrapolate from what we learned in Section 8.6.1 to generate the domain list for the simple network shown in Figure 1, but here it is:

```
# Site:           bar.com
# Administrator: Sneaky Pete (snookums@excite.com)
# Registrar:     dotster.com
.bar.com::ns.bar.com:3600
.bar.com::ns2.bar.com:3600
+*.bar.com:12.165.54.198:3600
@bar.com:12.165.54.198:bar.com::3600

# Site:           sneakers.org
# Administrator: Lone Gunman (webmaster@sneakers.org)
# Registrar:     dotster.com
.sneakers.org::ns.bar.com:3600
.sneakers.org::ns2.bar.com:3600
+*.sneakers.org:54.45.66.102:3600
@sneakers.org:54.45.66.102:sneakers.org::3600
```

Yup. Nothing real surprising there. We listed both the site we are hosting and the name we are serving for some other site. We simply included the other site's IP address instead of our own on the "+" and "@" records.

8.7. *Make*

That's about it for your vanilla version of djbdns. All that's left is to process the /services/tinydns/root/data file to build /services/tinydns/root/data.cdb:

```
$ cd /services/tinydns/root
$ make
```

The resulting data.cdb file may now be copied to any other tinydns install you have running. I recommend you use rsync to do this. If you're unfamiliar with rsync, you can learn more about it at http://rsync.samba.org/.

Also, /services/tinydns/root/Makefile is easily modified. I recommend you add an rsync command to the bottom of the file so that your name servers are automatically synchronized when you make changes to your DNS configuration.

9. Testing your configuration

If everything went well, you should now have DNS working on your machine. Do not just trust that it works. Instead, test it.

9.1. Test your resolver

If your resolver is working, then you should be able to look up domain names from your Linux box. Try the following experiment:

```
$ nslookup microsoft.com
Server:  cache1.yourdomain.com
Address:  192.168.167.11

Non-authoritative answer:
Name:    microsoft.com
Addresses:  207.46.230.220, 207.46.197.100,
207.46.197.102, 207.46.197.113
          207.46.230.218, 207.46.230.219
```

As you can see, nslookup (name server lookup) consulted the resolver I had listed in my resolv.conf (192.168.167.11). It also does a reverse DNS lookup on that address to get the resolver's name.

The resolver then looked up microsoft.com and listed all of their name servers.

The "non-authoritative answer" line indicates that the answer was pulled from our cache, and not directly from the server itself. Don't worry, that doesn't indicate a problem. If you are really anal, you could go ask one of microsoft's name servers directly:

```
$ nslookup
Default Server:  cache1.yourdomain.com
Address:  192.168.167.11

> server 207.46.138.20
Default Server:  dns1.cp.msft.net
Address:  207.46.138.20

> microsoft.com
Server:  dns1.cp.msft.net
Address:  207.46.138.20

Name:    microsoft.com
Addresses:  207.46.197.113, 207.46.230.218,
207.46.230.219, 207.46.230.220
          207.46.197.100, 207.46.197.102
```

As you can see, consulting their name server (dns1.cp.msft.net) directly yielded the same, but this time authoritative, results.

If you have multiple caches in your network, be sure to test them all. You can use the server command, like in the above example to specify which cache you wish to test. Hit control-D to exit from nslookup's interactive mode.

9.2. Test the internal DNS

Now that we know we can look up the domain names of other servers, let's find out if we can look up our own:

```
$ nslookup yourdomain.com
Server:  cache1.yourdomain.com
Address:  192.168.167.11

Non-authoritative answer:
Name:    yourdomain.com
Address:  192.168.167.20
```

As you can see, I ran the preceding test from within the UT region in Figure 3. tinydns recognized that the request came from an internal address, so it responded with an internal address.

9.3. Test the external DNS

If you have a firewall, then tinydns will respond with different IP addresses depending on whether requests are made from inside the firewall or outside. We already tested from within, but now we need to test from outside.

If you have ready access to a Linux box outside your firewall, then this is easy to do, just repeat the instructions in Section 9.2 from an external machine and look for your external IP addresses.

If you do not have ready access to another box, then you will need another box to make the request for you. The catch is that not every server will do this for you, so you may have to scout around a bit. If you find a server that does provide this server, don't abuse them by advertising that they do. This will just cause them to close down that service.

To find out if a server will do an nslookup for you, use nslookup in an interactive mode like we did at the end of Section 9.1, and ask them to look up a server that is known to be configured correctly (like microsoft.com). If the request times out then the server does not provide that service and you will have to try another.

When you do locate a server you can use, ask them to look up your domain name. If everything is configured correctly, you should get your external IP address returned:

```
$ nslookup
Default Server:  cache1.yourdomain.com
Address:  192.168.168.11

> server sound.net
Default Server:  sound.net
Address:  205.242.192.21

> yourdomain.com
Server:  sound.net
Address:  205.242.192.21

Name:    yourdomain.com
Address:  24.37.254.20
```

10. Reverse DNS

Well, I'm finally at the end of this mess and now it's time to talk about reverse DNS. Bleah. I hope all of my stalling hasn't made you think this part is going to be exciting. It's not.

In a nutshell, when they invented DNS, they decided they wanted a way to look up domain names given an address. They didn't, however, want to build another huge system like DNS to do it. I can't really blame them there. So they did what any good programmer would do and used the existing system to do both tasks.

As you have seen, a DNS search goes from general to specific. If you are looking for www.balloftwine.com, you start by finding a server that knows about "com". Typically, that points you to a server that knows about balloftwine.com, and that points you to a server that knows about www.balloftwine.com. It's more complex than that, obviously, but that's the general idea.

Domain names are sort of the reverse of an IP addresses. In the IP address 24.37.254.20, the most general part is 24, not 20. So to do the search, you look for a server that knows about 24, it points you to a server that knows about 24.37, and so on.

To make it so we could re-use the same system, when a computer does a reverse DNS lookup, they simply take the IP address, reverse the octets, tack on ".in-addr.arpa", and pretend it is a new domain name. To track down 24.37.254.20, the computer searches for a domain named "20.254.37.24.in-addr.arpa". Clever, huh?

Your name server, in essence, is programmed to answer questions about domain "254.37.24.in-addr.arpa" which includes the more specific "20.254.37.24.in-addr.arpa", just like balloftwine.com is more general than www.balloftwine.com.

It's strange, but at the same time, simple. So why did I save it to last to discuss it? Well, there's a lot of complications in registering your reverse DNS address -- a lot of complications you don't have in registering a domain name -- and I didn't want derail your train of thought while you were configuring tinydns.

So what *is* the complication? The complication is that there is no real system for registering a reverse DNS address like there is for registering a domain name.

If you rent a few IP addresses from an ISP (speakeasy.net, for example), they will respond to reverse DNS requests before the request is ever delegated to your box. Requests for 24.37.254.20 will net a response like dsl254-37-24.nyc1.dsl.speakeasy.net. In other words, it won't really matter what you have set up as your reverse DNS information, because the request won't come to you.

If you wish to do reverse DNS, then you must speak with your ISP. This is generally done by e-mail. Depending on your situation and ISP, one of the following will typically happen:

- If you rent a small number of IP's and you have good ISP, the ISP will typically enter your machine names (larry.yourdomain.com) into their DNS and do the reverse DNS resolution for you. This sort of thing is done at their leisure and will not happen instantly. Requests still won't make it to your box, but at least they will be answered correctly (eventually). Local requests will be handled by tinydns.

- If you rent a small number of IP's and you have a more typical ISP, your ISP will ignore your e-mails or reply that they'll do it, but then not really do anything. This will continue until you decide that reverse DNS isn't very important or you change ISP's. Feel free to kick up a storm, but be realistic about how much effort your business is worth to the typical ISP. Not much.

- If you rent a large block of IP's from a larger provider, you can typically get your provider to stop doing reverse DNS for your block. In this case, requests will actually make it to your machine and tinydns will be able to respond to reverse DNS queries.

11. Where to go from here

We'll that's about it for me. If you read everything in this book then you probably know as much as I do. If you still have questions, start surfing up http://cr.yp.to/djbdns.html again. Now that you've got a little more how-to under your belt it should be a little easier to understand what Professor Bernstein is talking about.

If that isn't enough, I would turn to http://www.lifewithdjbdns.org/. There's lots of good, user-contributed information there as well. It assumes you know than this book did, but hopefully you *do* know more now, so this won't be too bad.

If you still don't have enough information, the next place to turn is the djbdns mailing list. To subscribe, send a blank e-mail to dns-subscribe@list.cr.yp.to. Remember, the mailing list is manned by volunteers who use djbdns. They are very helpful, but they will probably assume you know more about DNS than you really do unless you state it clearly in your e-mail. Don't expect them to do the work for you, but they are good about answering clear questions.

Lastly, if you have something you think I should add to this book, please feel free to tell me. Keep in mind that this guide is aimed at newbie users and is only intended to get users from "How the heck do I set this up?" to "Yay! I'm online." It is not my goal to teach the advanced concepts of DNS or even learn them myself.

Finally, I ask that all suggestions to this book are e-mailed no later than 9/13/02. I wrote this book because I needed help when I went through the process myself, not because I wanted to spend my life maintaining a living document. I think six months should be plenty of time to work out the mistakes I've made and add examples to any sections where I have glossed over important details.

Good luck!

Gre7g Luterman (gre7g@wolfhome.com)