Running IPv6

Iljitsch van Beijnum

Apress®

Running IPv6

Copyright © 2006 by Iljitsch van Beijnum

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN : 1-59059-527-0

Library of Congress Cataloging-in-Publication data is available upon request.

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Jim Sumser
Technical Reviewers: Jordi Palet Martinez and Pim van Pelt
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Laura Cheu and Richard Dal Porto
Copy Edit Manager: Nicole LeClerc
Copy Editor: Kristen Imler
Assistant Production Director: Kari Brooks-Copony
Production Editor: Lori Bring
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: Linda Seifert
Indexer: Michael Brinkman
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, email orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at http://www.apress.com in the Source Code section.

CHAPTER 5

The DNS

"The author of the Iliad is either Homer or, if not Homer, somebody else of the same name."

—Aldous Huxley

For us humans, it's difficult and not very pleasant to work with IP addresses, especially with IPv6 addresses. Fortunately, the Domain Name System (DNS) allows us to work with much more user-friendly symbolic names most of the time.[1] Because the DNS translates from names to IP addresses for us (and the other way around), the DNS itself needs to be updated to support IPv6. Due to its distributed nature, making the Domain Name System IPv6-aware is much more complex than upgrading a single application. Later in this chapter, we'll look at the Berkeley Internet Name Domain (BIND) DNS server software, but before that, I'll provide a refresher on how the DNS works and the changes that were necessary (and the changes that were not so necessary) for IPv6. Figure 5-1 shows the interaction between different parts of the Domain Name System.

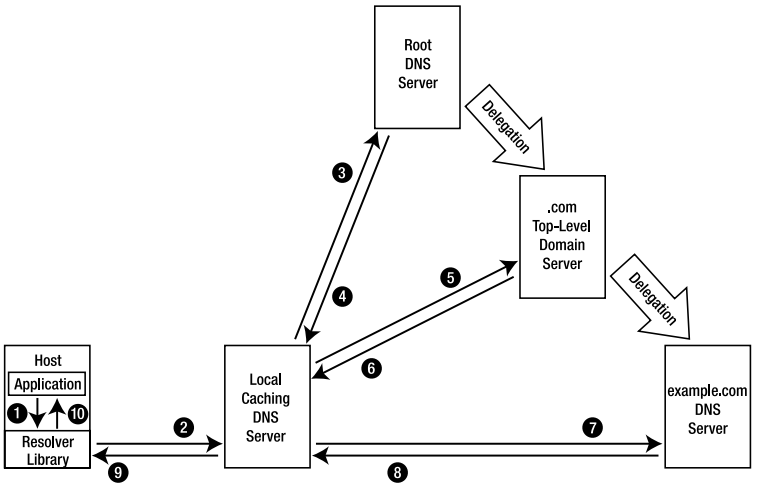

Figure 5-1. *Looking up information in the Domain Name System*

1. *DNS and BIND*, by Paul Albitz and Cricket Liu (O'Reilly & Associates), is an excellent guide to the subject. Unfortunately, the fourth edition was published in 2001, so it is outdated on the subject of IPv6 in the DNS.

When an application wants to communicate over the network, it takes the full name of the destination, such as `www.example.com` (this is often called the "fully qualified domain name," or FQDN),[2] and finds the matching IP address in the DNS. The application does this by calling on the resolver library. This is step 1 in Figure 5-1. The resolver library knows enough about the DNS protocol to be able to send a request for the required information to a "caching" or "recursive" DNS server using the DNS protocol (step 2). If the caching server just started and hasn't had the chance yet to live up to its name by caching the answers to previous requests in its memory, it will have no idea where to find the requested information. So it contacts one of the root DNS servers (step 3). The root servers don't know the address for `www.example.com` either, but they do have pointers to the DNS servers responsible for all "top-level domains" (TLDs), such as `.com`, so in step 4, the root server sends back pointers to the `.com` TLD nameservers. Information received from remote nameservers such as this is kept in the local cache if it was received from a server that's part of the authoritative delegation chain. The caching nameserver proceeds to contact one of the `.com` TLD servers and repeats the question about the address that goes with `www.example.com` in step 5. Like the root server, the TLD server doesn't know the answer, but it supplies pointers to the nameservers that are responsible for the `example.com` domain name in step 6. In step 7, the caching server once again asks for the address information for `www.example.com`, and this time the answer finally contains the requested information (step 8). The caching server can now send back a response to the resolver library in step 9, which in turn relays the information to the application (step 10). The application now has the information it needs to (for instance) set up a TCP connection to `www.example.com`.

Representing IPv6 Information in the DNS

Ignoring caching for a moment, every DNS request involves the root nameservers, TLD servers and the destination's nameserver, and often nameservers from the initiator's and/or destination's ISPs. Requiring all those nameservers to be upgraded to support IPv6 in order to be able to look up IPv6 addresses in the DNS would make it almost impossible to deploy IPv6. Fortunately, this isn't necessary. In 1995, RFC 1886 described a very straightforward way to publish IPv6 information in the DNS that provided an easy upgrade path. However, in 2000, a more ambitious way to do the same was published in RFC 2874. This new mechanism was partially implemented, but more detailed analysis and practical experience showed that it was perhaps a bit *too* ambitious, so around 2001, the IETF started moving away from the new method, and in 2003, RFC 1886 was reinstated for the most part but with one small, yet important change (RFC 3596).

■ **Note** All implementations of IPv6 reverse DNS lookups from before 2000 and many from before 2003 are outdated and may not work. If they work, they may not continue to do so for much longer.

---

2. Purists always type a period at the end of an FQDN, to indicate that it's an FQDN. This stops the resolver from thinking it needs to stick the local domain name to the end. In the `example.com` network, `www.kame.net` may mean `www.kame.net.example.com` or `www.kame.net`, but `www.kame.net.` is always `www.kame.net`.

RFC 1886: AAAA and ip6.int

In IPv4, addresses are stored in `A` (address) records, and the reverse mapping is done by creating a special domain name that consists of the values of the individual bytes in the address in reverse order, followed by `in-addr.arpa`. Looking up such a domain name results in a `PTR` (pointer) record containing the domain name associated with the address in question. See Listing 5-1.

Listing 5-1. *An IPv4 Address in the DNS*

```
www.example.com.          IN A   192.0.2.17
17.2.0.192.in-addr.arpa. IN PTR www.example.com.
```

As we'll see later in this chapter, these lines don't appear together in the DNS zone files.[3] The names end in a dot to indicate they are absolute addresses (FQDNs), so no additional domain names should be added. RFC 1886 sticks very close to the IPv4 way of doing things, as outlined in RFC 1035. IPv6 addresses are stored in `AAAA` ("quad A") records. The reverse mapping is done by taking the hexadecimal digits of the IPv6 address (all of them, including zeros that would normally be left out) in reverse order and adding `ip6.int`, as in Listing 5-2. When RFC was published in the mid-1990s, the `.int` level domain was the "infrastructure TLD" of choice, probably because at the time, it was the only truly international TLD; `.com`, `.edu`, `.gov`, `.mil`, `.net`, and `.org` were all still considered US-only, and `.arpa` was a relic from the by-then defunct ARPANET.

Listing 5-2. *An IPv6 Address in the DNS According to RFC 1886*

```
www.example.com. IN AAAA 2001:db8:1bff:c001::390
0.9.3.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.c.f.f.b.1.8.b.d.0.1.0.0.2.ip6.int. IN PTR ➥
www.example.com.
```

RFC 2874: A6, DNAME Bitlabels, and ip6.arpa

■ **Note** These mechanisms aren't in use on the Internet at present, so skipping this section is of little consequence.

By the turn of the millennium, basic IPv6 features were well established, so the IETF's attention focused on more subtle issues and on problems that IPv6 *didn't* solve. The biggest problem that IPv6 doesn't solve is routing, as we saw in Chapter 4. One thing would make the routing situation much better: rapid renumbering. If renumbering were trivial, people would be more likely to use address space that can be aggregated by ISPs so the routing tables stay smaller.

3. A "zone" is a "the complete database for a particular 'pruned' subtree of the domain space" (RFC 1035). A zone file contains all the DNS records for a domain and any subdomains that aren't delegated to another zone.

The A6 Name-to-Address Mapping

The RFC 2874 approach to renumbering is as follows. Suppose you ask someone for his work phone number, and the answer is "1-512-555-5501." This would enable you to call your friend, as long as nothing changes. Would the answer have been "extension 501 at IBM Research in Austin, Texas," you'd still be able to call (with a little extra effort), even if the actual number didn't work anymore, for instance, because the Austin area code changed. The RFC 1886 AAAA method is similar to simply giving out the requested number. RFC 2874 is more like providing a path through the addressing hierarchy, not unlike giving out a phone number by providing an extension, organization, and city. Listing 5-3 shows what an A6 hierarchy looks like in the DNS.

Listing 5-3. A6 *Records in the DNS*

```
www          IN  A6  64  ::0000:0000:0000:0390  subnet-a
subnet-a      IN  A6  48  0000:0000:0000:c001::  prefix-isp1
subnet-a      IN  A6  48  0000:0000:0000:c001::  prefix-isp2
prefix-isp1  IN  A6  0   2001:0db8:1bff::
prefix-isp2  IN  A6  0   3ffe:9500:003c::
```

Unlike the previous examples, all domain names in Listing 5-3 are relative. Assuming they're in the example.com zone file, the nameserver will add .example.com after each name. Every A6 record provides part of the address and a pointer to where the rest of the address can be found. The first A6 record (the one for www) leaves 64 bits blank to be defined later and continues to provide the IPv6 address ::390 to fill in the 128 – 64 = 64 bits that it does specify. The address bits specified in the A6 record at hand are copied from their respective place in the listed address. The part of the address that isn't specified here must be set to zero in the address provided in the zone file. The name following the IPv6 address-like value in the A6 record points to the place in the DNS hierarchy where the rest of the address can be found; in this case, under the name subnet-a. And indeed, under subnet-a is another A6 record or, rather, two of them. They both set the bits between 48 and 64 to c001 (the rest of the bits is zero) and point to prefix-isp1 and prefix-isp2, respectively, for the rest of the address. Under these names, the remaining bits from 0 to 48 are provided, and a pointer to elsewhere isn't needed, as the address is now complete. Because subnet-a has two pointers for the upper 48 bits, the whole procedure results in two complete addresses: 2001:db8:1bff:c001::390 and 3ffe:9500:3c:c001::390.

With A6 records, updating the DNS when there is a renumbering event is a breeze: rather than having to change all addresses in all domains for a site, only a single A6 record has to be changed. For instance, if the 3ffe:9500:3c::/48 prefix in Listing 5-1 were to be changed to 2007:4580:73::/48, this would only require an update of the prefix-isp2 record, an all A6 records that point to it automatically reflect the new information.

The Bitlabel and DNAME Address-to-Name Mapping

In addition to the A6 method for forward mapping, RFC 2874 also specifies a new way to do the reverse mapping from address to name. It uses two mechanisms that were defined in RFCs 2672 and 2673, respectively: DNAME and bitlabels. The DNAME record is somewhat similar to the CNAME record. But rather than providing an alias for a single name, like CNAME does, DNAME can provide an alias for an entire branch in the DNS tree: a domain or subdomain. Listing 5-4 shows DNAME in action.

Listing 5-4. *The* DNAME *Record*

```
research.example.com          IN DNAME  r-and-d.example.com.
www.plastics.r-and-d.example.com.  IN A      192.0.2.1
www.biotech.r-and-d.example.com.   IN A      192.0.2.2
```

With this DNAME record in effect, all records and subdomains under r-and-d.example.com are also present under research.example.com. So looking up www.biotech.research.example.com has the same result as looking up www.biotech.r-and-d.example.com. To be backward compatible, the nameserver will "synthesize" a CNAME record for the requested information, along with providing the actual DNAME record. However, apparently some older resolver libraries wouldn't handle DNAME records properly.

The idea behind bitlabels (also sometimes called "binary labels") is that the traditional ...4.3.2.1.in-addr.arpa or ...e.f.f.3.ip6.int delegation mechanism is less than perfect because it only allows delegation on 8- or 4-bit boundaries, respectively. So conceptually, a bitlabel is an expression of a very long domain name with individual bits separated by periods. (In the domain name system, the data between two periods is called a "label.") However, within the DNS protocol, a bitlabel is expressed as a single chunk of binary data, regardless of the number of bits it contains, rather than a long list of individual ASCII labels. In the DNS zone files, bitlabels may be specified in either binary, octal, decimal, or hexadecimal, with an explicit value indicating the length in bits. Listing 5-5 shows several bitlabel representations of the same information.

Listing 5-5. *Bitlabels in Zone Files*

```
\[xf0d2b496785a3c1e]               IN PTR www.example.com.
\[b11110000110100101011010010010110011110000101101000111100000011110] IN PTR ➥
www.example.com.
\[o741512644547413217017O/64]      IN PTR www.example.com.
\[120.90.60.30].\[240.210.180.150] IN PTR www.example.com.
```

The first bitlabel is in hexadecimal, as denoted by the initial x. The second is in binary (b) and the third is in octal (o). Because one octal digit represents three bits, the 22-character string would normally specify 66 bits. The explicit /64 indicates that only 64 bits should be considered part of the bitlabel. The last line is in dotted-quad decimal notation. Because this notation is limited to 32 bits, we need to concatenate two bitlabels to arrive at the full 64 bits. Note that although *within* each bitlabel the more natural most-significant-to-least-significant notation is used, the domain name system's least-significant-to-most-significant label ordering comes back when concatenating bitlabels. So if 64-bit decimal dotted-quads were allowed, that version of the bitlabel in Listing 5-5 would look like \[240.210.180.150.120.90.60.30]. Together, DNAME and bitlabels allow reverse mapping information to be delegated as in Listing 5-6.

Listing 5-6. *Reverse Mapping with* DNAME *and Bitlabels*

```
\[x20010DB81BFF/48].ip6.arpa.  IN  DNAME  rev.example.com.
\[xC001/16].rev.example.com    IN  DNAME  srvrs.rev.example.com.
\[x0000000000000390/64].srvrs.rev.example.com. IN PTR www.example.com.
www.example.com.        IN  AAAA   2001:db8:1bff:c001::390
```

In real life, the first line would have to be a delegation by an ISP, so it would be in the ISP's zone file. However, the other lines could all be in the same zone file (the one for `example.com`, for instance), or they can be spread out across several zones for added flexibility and ease of renumbering.

RFC 1886 vs. RFC 2874

When a host that implements RFC 1886 looks up an IPv6 address in the DNS, its resolver library will send out a request for `AAAA` records. Obviously, the last DNS server in the chain must understand what those are, but the intermediate DNS servers (see Figure 5-1) don't; they just see a resource record type that they don't recognize, but the format is familiar so they know how to process the information. Looking up reverse information in the `ip6.int` domain is even easier, because to a nameserver, there is nothing special about this domain.

The same is true for processing `A6` records: this is all done by the resolver, so again, nameservers in the middle don't have to understand the `A6` semantics. However, obtaining an address this way is a fairly involved process, especially if the pointers from one `A6` record to the next jump between different servers in different domains, or when the chain of `A6` pointers is very long. The fact that this procedure is supposed to be executed by the resolver and not the caching nameserver necessitated a substantial rewrite of the BIND software and the addition of a resolver daemon. The traditional resolver library wasn't really equipped to handle such complex tasks.

Although full support for the `DNAME` record requires changes to caching nameservers as well as to the nameservers hosting the `DNAME` information, the synthesis of additional `CNAME` records makes it possible for unmodified resolvers and caching nameservers to work with `DNAME`. Things are different for bitlabels, however. DNS queries that contain bitlabels need different processing from queries that only contain traditional ASCII labels, so in addition to the resolver and the nameserver holding the bitlabel information, *all* nameservers in between (the caching nameserver, root, and TLD servers) must understand bitlabels.

RFC 3596: AAAA and ip6.arpa

Not surprisingly, there was considerable debate in the IETF over the relative merits of the RFC 1886 and the RFC 2874 ways of doing things. Part of this discussion condensed in RFCs 3363 and 3364 in 2002. The main arguments in favor of RFC 2874 were flexibility and support for rapid renumbering. The arguments against using `A6` records to store IPv6 addresses in the DNS, and bitlabels to perform reverse mapping, were that they add complexity and increase the time needed for looking up the information (if it's spread out over several nameservers). RFC 3364 also notes that `A6` records are "optimized for write" even though "reading" DNS information is much more frequent than changing it. It's also hard to imagine a way in which the information in the DNS would remain in sync with the actual addresses used by hosts during a renumbering event.

Eventually, this led to the conclusion that `AAAA` records would be the best way to store IPv6 addresses in the DNS, and the nibble method the preferred way to do reverse mapping. A nibble is 4 bits, which refers to the `...1.0.0.2.ip6...` reverse mapping technique. However, in the mean time, in 2001, the Internet Architecture Board (IAB) had published RFC 3172, stating a preference toward `.arpa` (now "Address and Routing Parameter Area") as an infrastructure top-level domain, complicating a complete return to RFC 1886 and `ip6.int`. Subsequently, use of `ip6.int` was "deprecated" in favor of `ip6.arpa` in Best Current Practice (BCP) document 49, also known as RFC 3152. All of this culminated in RFC 3596 (2003), which standardizes the use of `AAAA` records and the nibble method for reverse lookup under `ip6.arpa`.

The Current Situation

After such turmoil, it's not entirely surprising that implementations were all over the map. Although the `A6` record never gained much traction, all the different ways of looking up reverse mapping can still be found "in the wild." A few rather old IPv6 resolver implementations use `ip6.int` exclusively. This isn't a huge deal, as many people set up both `ip6.int` and `ip6.arpa` (nibble method). And if the `ip6.int` lookup fails, it usually does so with a regular "no such domain" error message, which doesn't lead to additional problems. The IETF seems to want to get rid of `ip6.int` in a hurry, after which `ip6.int`-only implementations won't be able to resolve IPv6 addresses into domain names.

A larger group of implementations (including many versions of Red Hat Linux) looks for bitlabels under `ip6.arpa` first, and then falls back to the nibble method under `ip6.int`. Because there is no evidence of there ever having been any bitlabel delegations, the first step will always be unsuccessful, even if all the caching nameserver and the `ip6.arpa` TLD nameservers all understand bitlabel queries. This may not be the case, however, as most of the available DNS server software doesn't support bitlabels. BIND gained support for bitlabels around version 9, but it was removed again in version 9.3.

Then there are some resolver implementations that first look for `ip6.arpa` using the nibble method, and if they don't find anything, fall back on `ip6.int`, and finally, there are the implementations that only look for `ip6.arpa`. A similar spectrum of behavior is present in the DNS utilities `dig`, `host`, and `nslookup` that are part of the BIND distribution. Because these contain their own resolver code, their behavior may or may not match that of the rest of the system.

IPV6 AND THE ROOT SERVERS

For nameservers to resolve information over IPv6, it's necessary that the root nameservers gain IPv6 support. Nameservers find the root DNS server addresses in a local "hints" file, so it would appear that providing the roots with IPv6 connectivity and updating the hints file would do the trick. Unfortunately, there are some complications. Because hints files tend to get out of date, the first thing a nameserver does upon startup is ask one of the nameservers listed in the hints file for the current list of root nameservers. So, for nameservers to be able to communicate with the root nameservers over IPv6, the answer to this initial query must contain IPv6 addresses for at least a subset of all root servers.

In theory, it's fairly trivial to add IPv6 addresses for the root nameservers as "glue" records in the root zone. The problem is that the original DNS specifications only allow for 512 byte DNS messages over UDP. For the current list of 13 root servers, the initial response message is 436 bytes (and that's after "label compression" to avoid repeating the same domain name for different servers), so there is no room to add IPv6 addresses for all root servers without potentially going over the limit. When that happens, some addresses must be dropped from the response, which often means the request must be repeated over TCP. Because many people are unaware that regular DNS queries and not just zone transfers can happen over TCP, this is often filtered in firewalls. RFC 2671 adds support for larger DNS messages through the "EDNS0" mechanism, but a sizeable minority of all nameservers on the Internet don't support EDNS0.

Since mid-2004, TLD registries may have IPv6 addresses included in the root zone as glue records, and some TLDs allow end users to register IPv6 nameserver addresses for their domains. Many of the root nameservers are already reachable over IPv6 (see `http://www.root-servers.org/`). ICANN and the root server operators are proceeding very cautiously, but addition of IPv6 glue records to the root zone is expected in the not too distant future.

Installing and Configuring BIND

On most UNIX-like systems there is no need to install BIND, as there is generally a BIND 9.x (often 9.2.x) included with the system. (FreeBSD 4.x comes with a recent release of BIND 8.x, but BIND 9 is available in the ports collection.) BIND consists of a number of core programs and supporting utilities. The main program is the `named` binary. This is the actual nameserver daemon.

■ **Note** Invoke `named -v` to determine the BIND version installed on the system. You can generally find out the BIND version of a remote nameserver by asking for the `TXT` record in class "chaos" for the domain name `version.bind`: `host -c chaos -t txt version.bind <nameserver name/address>`.

Installing BIND

Different versions of BIND are available from the Internet Systems Consortium (ISC, formerly Internet Software Consortium) at `http://www.isc.org/sw/bind/`. For full IPv6 support, including IPv6 transport (answering queries received over IPv6), you should choose one of the 9.x versions; 9.2.x for bitlabel and `A6` support. However, all the latest releases of currently maintained versions (including 4.9.x) support `AAAA` records, as do all other major DNS server implementations. A binary distribution of BIND is available for Windows, which runs on Windows NT, 2000, XP, and 2003. This distribution doesn't support IPv6 transport yet. If you want to install BIND on a Windows machine anyway, read the `readme1st.txt` document carefully. Obviously, the locations of the files discussed below will be different, but otherwise, BIND under Windows is pretty much the same as BIND under a UNIX-like operating system.

Should you wish to install a different version of BIND than the one that came with your Linux or FreeBSD system,[4] just download the source and go through the customary `./configure`, `make`, `make install` sequence per the instructions in the `README` file. Execute the `configure` script with the argument `-h` to list available options. Getting BIND to compile under MacOS requires advanced UNIX hacking skills: BIND doesn't get along very well with the changes that Apple made to the UNIX/FreeBSD system core.

Starting BIND at Boot Time

Under Red Hat Linux, there is already a startup script for `named`, but it's not activated by default. This can be done by issuing (as root) the `chkconfig named on` command to create the necessary symbolic links to the `/etc/init.d/named` script. Unsurprisingly, `chkconfig named off` removes the links and stops `named` from being initiated at system startup. `chkconfig --list` shows which startup scripts will be executed when changing run levels.

Under FreeBSD, starting `named` at boot time is done by adding two lines to `/etc/rc.conf`, as shown in Listing 5-7. Obviously, it helps if the `named` daemon is indeed located in `/usr/sbin/`.

---

4. If you're upgrading from a 4.x version of BIND, install an 8.x version first, as the 8.x distribution contains tools to convert 4.x configuration files to the format used by BIND 8.x and later.

Listing 5-7. *Enabling* named *in* /etc/rc.conf *Under FreeBSD*

```
named_program="/usr/sbin/named"
named_enable="YES"
```

Configuring BIND

All of BIND's extensive configuration options are described in the BIND Administrator Reference Manual that comes with the source or binary distribution. BIND isn't hard to run: just typing named starts the daemon. Sending named the HUP signal will make it reload its configuration and zone files. Because named needs access to TCP and UDP port 53, it must be run (at least initially) as root. Sending the HUP signal must be done under the same user as the one named runs under (or as root), which is often inconvenient. An alternate method to control the nameserver is with the remote name daemon control program rndc. rndc connects to named over TCP, so the command can also be used to control remote nameservers, as the name suggests. By default, named listens for incoming connections from rndc on port 953 on the IPv4 and IPv6 localhost addresses only, but this can be changed with the controls configuration command in the named configuration file. rndc expects a configuration file in /etc/rndc.conf, but it's much easier just to execute rndc-confgen -a to create an /etc/rndc.key file, which both rndc and named will then use to authenticate the communication between them.

The nameserver files are often stored in /var/named, but there is no particular reason to adhere to this convention. The named directory and other files must be readable and writable as appropriate for the user that named ends up running under. Because named drops most special root privileges, including the ability to access other user's files, when it runs as root, the files must be accessible to the actual user root itself in this case. The named directory must contain a named.root file, which guides named toward the root servers at startup. This file doesn't change often,[5] and as long as there is still a single correct root server address in it, named will be able to get an up-to-date list of root server addresses from that server. However, if your version is older than January 29, 2004, you may want to download the latest version from ftp://ftp.internic.net/domain/named.root or http://www.iana.org/popular.htm. For an IPv6-only nameserver to be able to reach the root servers, a new named.root needs to be installed when AAAA records for the root servers are added to the root zone.

The location of the named directory and the named.root file must be listed in named's configuration file. Listing 5-8 shows a basic named.conf file.

Listing 5-8. *The* /etc/named.conf *File*

```
options {
  directory "/var/named";
  allow-recursion { 192.0.2.0/24; 2001:db8:1bff::/48; };
  listen-on { 192.0.2.106; }
  listen-on-v6 { any; };
```

5. MacOS X Panther comes with a named.root (under the name "named.ca") from 1997, but only two root nameservers have changed addresses between the 1997 and 2004 versions.

```
# forward first;
# forwarders { 192.0.2.53; };
/* C-style comment */
// C++-style comment
};

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

zone "0.0.127.IN-ADDR.ARPA" {
  type master;
  file "localhost.rev";
};

zone "example.com" {
  type slave;
  file "example.com";
  masters { 192.0.2.53; };
};

zone "0.0.0.0.f.f.b.1.8.b.d.0.1.0.0.2.ip6.arpa."
  {
    type master;
    file "db.2001:db8:1bff:0";
  };
```

The file starts with an `options` directive, followed by several options between braces. Semicolons terminate statements or items in a list. The first option specifies the directory where `named` looks for files. The `allow-recursion` option defines for which clients `named` will perform recursive queries. In this case, it's clients with addresses in prefixes `192.0.2.0/24` and `2001:db8:1bff::/48`. Although there is no direct harm in allowing recursive queries for the whole Internet, and it allows for easier debugging, it can cost extra bandwidth, processing overhead and memory if the rest of the Internet starts using your server en masse. Many of the security problems found in BIND over the years could only be exploited by people for whom the server would do recursive queries. If you want to limit certain things to localhost-only, be aware that the keyword `localhost` only means the IPv4 localhost address in the named.conf file. The IPv6 localhost address must be listed explicitly as `::1`, if desired.

The `listen-on-v6` option directs `named` to listen for queries on IPv6 TCP and UDP sockets. With BIND 9.2 and earlier, `listen-on-v6` only takes "any" or "none" as arguments. So either the server will listen for incoming queries on all IPv6 addresses, or it won't listen on IPv6 at all. The default is to not listen on IPv6 addresses. As of BIND 9.3, you can have `named` listen on specific IPv6 addresses.

■ **Note** Specifying `listen-on-v6 { none; }` will *not* disable the use of IPv6 altogether. The server will still perform queries of its own over IPv6 when it deems necessary. It's also still possible on a server that acts as a slave (secondary) server for a zone to specify IPv6 addresses for primary masters to load the slave zone from.

The next two lines are commented out. In addition to these shell-style comments, `named` also accepts C and C++ style comments, but it doesn't accept the semicolon as the start of a comment, like in a zone file. The two initial commented-out lines would have instructed the server to forward all its queries to the nameserver on address `192.0.2.53` and only try to resolve the query on its own if the specified server doesn't reply. After the comments, the closing brace ends the `options` section. Next, there are four zone specifications:

1. The "dot" zone (the root) is a "hints" zone and points to the `named.root` file.
2. The `0.0.127.in-addr.arpa` zone is the reverse zone for the localhost address, and, being a primary zone, authoritative information is present in the localhost.rev file.
3. The `example.com` zone is a slave zone, and authoritative data is periodically transferred from the nameserver at address `192.0.2.53` and stored in the `example.com` file.
4. The last zone is a nibble-style `ip6.arpa` zone for `2001:db8:1bff::/48`.

■ **Tip** In most service provider or larger enterprise environments, it's generally a good idea to have different authoritative and resolving DNS servers so that problems with one type of DNS server don't impact the other type.

BIND 9.2 supports an `allow-v6-synthesis` option, which will take `A6` records and bitlabel reverse mapping information and turn those into `AAAA` records and nibble-style `ip6.int` reverse mapping information. This option was introduced to ease the transition from RFC 1886 to RFC 2874, but now that RFC 2874 has fallen out of grace, the option has been removed again from BIND 9.3. The argument to this option is a list of addresses for which this synthesis is performed. Synthesis is only performed for recursive queries.

BIND 9.3 removes support for bitlabels. Zone files with bitlabel definitions are rejected. Version 9.3 still supports `A6` records to some degree; those may be present in zone files and will be included in responses when appropriate. However, unlike BIND 9.2, BIND 9.3 never uses `A6` records to find IPv6 address for other servers when following a delegation chain. New in BIND 9.3 is the `dual-stack-servers` option, which takes one or more names or addresses as its argument. When an IPv4-only `named` can't resolve a query because it needs IPv6 connectivity, or an IPv6-only `named` because it needs IPv4-connectivity, it will consult the server listed under `dual-stack-servers`. This server is supposed to have dual stack IPv4+IPv6 connectivity, so it should be able to perform all possible queries. This option helps BIND deal with a fragmented namespace.

NAMESPACE FRAGMENTATION

If a domain is served by nameservers that only have IPv6 addresses, names under that domain can't be resolved by nameservers that only have IPv4 connectivity. Conversely, IPv6-only nameservers can't resolve names that are only served by IPv4 nameservers. The situation where the domain namespace looks different depending on the IP version used can lead to problems. For instance, suppose a mailserver is only reachable over IPv6, and the nameserver pointing to this mailserver is also only reachable over IPv6. Someone in the IPv4-world will never be able to deliver mail to this server, but worse, they wouldn't even be able to see that the (sub-) domain in question exists in the first place. So, rather than sending back a "message couldn't be delivered" error, the mailserver sends back "domain doesn't exist," which is much more destructive, especially when the lack of connectivity for an IP version is only temporary.

For this reason, it is *highly* recommended that, for the foreseeable future, all DNS zones be served by at least one nameserver (preferably two) with an IPv4 address, even if the zone just contains IPv6-only information.

Choosing an Address for Your Nameserver

Specifying nameservers by name doesn't usually work: if we knew which addresses went with which names, we wouldn't have to consult a nameserver in the first place. So nameserver addresses tend to find their way to lots of different places:

- The address for a caching nameserver will be in lots of `/etc/resolv.conf` files (or the equivalent on other operating systems).
- The address for primary and secondary DNS servers for a domain will be listed in the TLD zone for that domain.
- The address for a primary DNS server will be in the `named.conf` of secondary servers.
- Often, the address for secondary DNS servers for local primary zones will be listed as addresses the server will accept zone transfers from.

All of this means that it's a good idea if nameserver addresses are as stable as they can be, and it doesn't hurt if they're easy to remember, either. So using EUI-64-derived IPv6 addresses for nameservers isn't the best idea, as the address will change whenever a network interface card is replaced or the DNS service is moved to another server. So a manually specified address is the best choice. Additionally, it's not a bad idea to put this address in a `/64` of its own. This way, it's easy to move the DNS address around the network. It's helpful to use the `transfer-source-v6` and `notify-source-v6` options to set the source address for outgoing zone transfer requests to the appropriate address.

Adding IPv6 Information to Zone Files

Before you fire up your favorite text editor and start adding `AAAA` records to all your zone files, you should first consider the implications. When a host's IPv6 address is listed in the DNS, IPv6-capable applications on other IPv6-capable hosts will generally prefer to connect over IPv6 rather than IPv4. When connecting over IPv6 doesn't work, the application may or may

not fall back on IPv4. Unfortunately, IPv6 connectivity is still often slower and less reliable than IPv4 connectivity. On the other hand, the only people who'll suffer when IPv6 performance is worse than IPv4 performance are those who enabled IPv6 on their end in the first place, so there is no need to be overly conservative. In most situations, there shouldn't be too many problems advertising an IPv6 address in the DNS, but for certain critical services, it can be better to provide the service over IPv6 under a different name. This is especially true if the applications used to access the service don't automatically fall back on IPv4, or when the timeout when this happens is unacceptable. It never hurts to have the service available under a different name that only has an IPv4 address. For file downloads over HTTP or FTP, it's a good idea to explicitly list IPv4 and IPv6 addresses, so people can choose, as file transfer is one of the applications that is most vulnerable to bandwidth limitations.

When adding `AAAA` records for popular services to your DNS zone, make sure that you can handle the additional IPv6 traffic. For services that are mostly used over the Internet, the amount of IPv6 traffic relative to IPv4 traffic isn't likely to be a problem. However, if you're upgrading an internal network, it's possible that all the internal hosts that now have IPv6 connectivity connect to the servers that now have an `AAAA` record over IPv6, effectively moving *all* traffic for the application in question from IPv4 to IPv6 over night. Most routers can handle IPv4 and IPv6 equally well, but some can't. For instance, when Cisco rolled out IPv6 on its large Cisco 12000 routers, IPv6 forwarding was done on the CPU on the linecard. Because those linecards had special IPv4 forwarding hardware on board, the CPUs on these linecards weren't designed to do forwarding at line rate. This meant that a Gigabit Ethernet linecard could handle 1000 Mbps IPv4 traffic, but only some 100 Mbps of IPv6 traffic (or 900 Mbps IPv4 traffic plus 100 Mbps IPv6 traffic). Newer linecards also have IPv6 hardware support so they can handle IPv6 at line rate, but you may still encounter hardware that doesn't handle IPv6 at the same speeds as IPv4. On Cisco routers with "smart" linecards, the linecard that receives a packet is the one that must do the processing required to forward the packet, and the capabilities of the outgoing card are less of an issue. Be sure to do your homework before buying new routers or before making changes to your network that may create a lot of IPv6 traffic.

If the existing servers can't handle IPv6, it may be necessary to set up a different server or cluster of servers to provide an existing service over IPv6. Then, you point one or more `A` records to the IPv4 server or servers, and one or more `AAAA` records to the IPv6 server or servers. However, be careful that you use *service* names and not *machine* names when you do this. So if the service name `www.example.com` points to the machine `zeus.example.com`, which provides the IPv4 WWW service, you should give the new IPv6 WWW server its own name, and not add an `AAAA` record to `zeus.example.com`. That way, you won't end up on `poseidon.example.com` when you type `ssh zeus.example.com` because SSH prefers IPv6.

AAAA Records

Listing 5-9 shows a zone file with `AAAA` records used differently for different services.

Listing 5-9. *A Zone with* `AAAA` *Records*

```
;   20041215    IvB created
;   20050209    IvB added AAAA records

$TTL 86400
```

```
@   IN  SOA ns1.example.com. root.example.com. (
              2005020900      ; Serial
              28800           ; Refresh (8 hours)
              7200            ; Retry (2 hours)
              604800          ; Expire (7 days)
              86400 )         ; Minimum (1 day)

              IN  NS      ns1.example.com.
              IN  NS      ns2.beispiel.de.

              IN  MX      100 smtp.example.com.
              IN  MX      200 smtp.ipv4.example.com.

              IN  A       192.0.2.80
              IN  AAAA    2001:db8:31:1:201:2ff:fe29:2640

ns1           IN  A       192.0.2.80
              IN  AAAA    2001:db8:31:53::53
              IN  A6      0 2001:db8:31:53::53

www           IN  A       192.0.2.80
www           IN  AAAA    2001:db8:31:1:201:2ff:fe29:2640
www.ipv4      IN  A       192.0.2.80
www.ipv6      IN  AAAA    2001:db8:31:1:201:2ff:fe29:2640

smtp          IN  A       192.0.2.25
smtp          IN  AAAA    2001:db8:31:1:20a:95ff:fecd:987a
smtp.ipv4     IN  A       192.0.2.25

pop           IN  A       192.0.2.25
popv4v6       IN  A       192.0.2.25
popv4v6       IN  AAAA    2001:db8:31:1:20a:95ff:fecd:987a
```

The file starts with two lines of comments. The `$TTL` line defines a default time to live value of 86400 seconds (one day), so all records that don't have their TTL set explicitly are cached for 24 hours. The next line starts with an at sign, which means that the record that follows relates to the zone itself rather than a name under that zone. The "start of authority" (`SOA`) record first lists the name for the primary nameserver for this zone (`ns1.example.com`) and a contact email address with the at sign replaced by a period (the email address `root@example.com`). The `SOA` record continues on the next five lines until the closing parenthesis. The first of those lines contains the serial number, which must be increased whenever a zone file is changed. It's customary to use a YYYYMMDDNN (year, month, day, number) format, where NN is set to zero when the first change on a certain day happens and increased by one on subsequent changes. The other lines define some timers that are best left unchanged.

There is no name in front of the `NS` records that follow, which means that they apply to the same name as did the previous record, which in this case is the domain itself. The primary nameserver for this domain is listed first, although ordering has no meaning within a zone file. Next are the `MX` records, which define where the mail for this domain should go to. The value

preceding the mailserver names indicates which mailserver is preferred. In this case, `smtp.example.com` is preferred over `smtp.ipv4.example.com` because it has a lower preference value. Both names point to the same IPv4 address, but `smtp` also has an IPv6 address. Should a remote mailserver have trouble with `smtp.example.com`, it will automatically fall back on `smtp.ipv4.example.com`. The next two lines supply `A` and `AAAA` records for the domain itself. These are useful when someone tries to connect to the domain itself, for instance, with the URL `http://example.com/`.

The next three lines define the addresses for `ns1.example.com`. Even though regular applications don't look for `A6` records, BIND versions 9.x prior to 9.3 (which are still in wide use) do, so supplying the IPv6 address of the nameserver in this format could speed things up a bit. The `AAAA` record is more important because it's the official way to publish an IPv6 address.

The `www` name has both an IPv4 and an IPv6 address and is supplemented by IPv4- and IPv6-only versions (`www.ipv4` and `www.ipv6`, respectively). Having an IPv6-only name is useful for quick IPv6 reachability tests: if the page loads, IPv6 is enabled and it works. If it doesn't, IPv6 either isn't enabled, or there is no connectivity. The addresses for the `smtp` and `smtp.ipv4` names reflect the earlier discussion. Finally, because the POP service is a critical one, and it's difficult for email users to temporarily change the address for their POP server when there is an IPv6 connectivity problem, the name corresponding to this service only has an IPv4 address. However, there is an alternate name `popv4v6` with both an IPv4 and an IPv6 address for users who prefer to use IPv6 when available, while maintaining the ability to fall back to IPv4.

Before the new domain can be used, it must be added to the `named.conf` file as in Listing 5-8. It's a good idea to check the syntax of the zone file and the configuration file before reloading the nameserver, like in Listing 5-10.

Listing 5-10. *Checking a Zone and Configuration Files and Reloading* `named`

```
# named-checkzone example.com /var/named/example.com
zone example.com/IN: loaded serial 2005020900
OK
# named-checkconf
# rndc reload
```

Reverse Mapping

The reverse mapping zones are by and large the same as regular zones, except that they contain only `PTR` records, except for the initial `SOA` and `NS` records. However, the nibble format is somewhat abrasive. The easiest way to turn IPv6 addresses into nibble format is by using the `host` command to look up the addresses in question. `host` will then echo back the nibble format query that it performs, which can then be copied and pasted into the zone file. Listing 5-11 shows the `host` command and Listing 5-12 the resulting reverse zone file.

Listing 5-11. *The* `host` *Command*

```
host 2001:db8:31:1:201:2ff:fe29:2640
Host 0.4.6.2.9.2.e.f.f.f.2.0.1.0.2.0.1.0.0.0.1.3.0.0.8.b.d.0.1.0.0.2.ip6.arpa not ➥
found: 3(NXDOMAIN)
```

If the `host` command tries to perform a bitlabel query, the program came with a BIND version 9.x prior to 9.3. Use `host -n` instead, and it will do an `ip6.int` query.

Listing 5-12. *A Reverse Zone*

```
;   20050209    IvB     created

$TTL 86400

@   IN  SOA ns1.example.com. root.example.com. ( 2005020900 28800 7200 604800 ➥
86400 )

            IN  NS      ns1.example.com.
            IN  NS      ns2.beispiel.de.

$ORIGIN 3.5.0.0.1.3.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
; ns1       IN  AAAA    2001:db8:31:53::53
3.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0  IN  PTR     ns1.example.com.

$ORIGIN 1.0.0.0.1.3.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
; www       IN  AAAA    2001:db8:31:1:201:2ff:fe29:2640
0.4.6.2.9.2.e.f.f.f.2.0.1.0.2.0  IN  PTR     www.example.com.

; smtp      IN  AAAA    2001:db8:31:1:20a:95ff:fecd:987a
a.7.8.9.d.c.e.f.f.f.5.9.a.0.2.0  IN  PTR     smtp.example.com.

$ORIGIN 1.3.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
0.0.2.c     IN  NS      ns.research.example.com.
```

The `$ORIGIN` directive specifies which location in the DNS hierarchy the names that follow are relative to. It's easiest to specify the top 64 address bits in a `$ORIGIN` line and then specify the remaining 64 bits on each individual line. This makes for one `$ORIGIN` per subnet and keeps the lines from becoming overly long. It can also be useful to keep the `$ORIGIN` statements out of the zone file and let all names in the file be relative, so the same zone file can be used for both the `ip6.arpa` and `ip6.int` zones.

This file holds the zone for prefix `2001:db8:31::/48`, which corresponds to the `1.3.0.0.8.b.d.0.1.0.0.2.ip6.arpa` zone. Information for two `/64`s under this `/48` is available in this zone file itself, but `2001:db8:31:c200::/64` is delegated to the nameserver `ns.research.example.com` in the last line.

■ **Note** Remember that there are 32 nibbles in an IPv6 address, so there should be 32 hexadecimal digits separated by dots in a full `PTR` record, 16 in a `/64`, and 12 in a `/48` `$ORIGIN`, respectively.

RFC 1886 and 2874 Reverse Mapping Hacks

Some resolver libraries fail in ugly ways because the bitlabel ip6.arpa information that they're looking for doesn't exist. To avoid these problems, you may want to set up fake reverse mapping information for them. This is done in Listings 5-13 (the bitlabel zone) and 5-14 (the relevant part from the named.conf).

Listing 5-13. *Fake RFC 2874 Reverse Mapping*

```
$TTL 86400

@   IN  SOA ns1.example.com. root.example.com. ( 2005020900 28800 7200 604800 ➥
86400 )
                    IN  NS      ns1.example.com.

*.\[x2/3].ip6.arpa. IN  PTR     bit.label.ip6.arpa.

\[x20010db800310001020a95fffecd987a/128].ip6.arpa. IN CNAME a.7.8.9.d.c.e.f.f.f.5. ➥
9.a.0.2.0.1.0.0.0.1.3.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
```

Listing 5-14. *Fake RFC 1886 and 2874 Zones in named.conf*

```
zone "\[x2/3].ip6.arpa." {
  type master;
  file "bitlabel.ip6.arpa";
};

zone "ip6.int." {
  type master;
  file "ip6.int";
};
```

The *.\[x2/3].ip6.arpa. is a wildcard that matches all domains (including bitlabels) under the three-bit bitlabel for 2000::/3, the IPv6 global unicast address space. Using 2000::/3 rather than ::/0 is a bit of a hack, but it avoids conflicts with possible real delegations directly under ip6.arpa, because the ::/0 bitlabel delegation would also have to be directly under ip6.arpa. Because the bitlabel space isn't delegated, there is no risk of this there. The other bitlabel at the end of the zone file matches only a single address and redirects the bitlabel version of this address to the nibble version with a CNAME record. This is useful to have the actual reverse mapping information show up for a limited set of addresses (which must all be listed individually). The wildcard record matches all addresses but has the disadvantage that the same reverse mapping information (bit.label.ip6.arpa in this case) is returned for all possible IPv6 addresses, except the ones that have an individual listing.

Listing 5-14 also has a delegation for the ip6.int domain, which can be useful when the real ip6.int zone is taken out of commission. The DNAME record in Listing 5-15 can then be used to redirect ip6.int queries to ip6.arpa. But as long as ip6.int is still active, it's better to do nothing and have the real ip6.int information show up.

Listing 5-15. *Remapping* `ip6.int` *to* `ip6.arpa` *Using a* `DNAME` *Record*

```
$TTL 86400

@   IN  SOA ns1.example.com. root.example.com. ( 2005020900 28800 7200 604800 ➥
86400 )
        IN  NS      ns1.example.com.

@       IN  DNAME   ip6.arpa.
```

Dynamic DNS Updates

RFC 2136 introduced the concept of "dynamic DNS updates." This mechanism allows a client to ask an authoritative server to add information to a zone or delete existing information from the zone. The dynamic update mechanism allows hosts that receive a new address through DHCP or stateless autoconfiguration to update their own DNS records so they remain reachable by their name, despite address changes. However, this capability isn't yet widely implemented in host operating systems, at least not for IPv6.

For obvious reasons, it's not possible for just any client to modify any and all zones. BIND 9.x accepts a zone file configuration option `allow-update` that lists who may update the zone. The list of authorized users may be in the form of IP address ranges, or it may specify one or more keys that protect the updates. See the BIND documentation for more information. When a zone is set up for dynamic updates, `named` takes control of the zone file and it's no longer possible to edit the file without first shutting down the `named` daemon.