Pro DNS and BIND

Ron Aitchison

Apress®

Pro DNS and BIND

ISBN (pbk): 1-59059-494-0

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

Lead Editor: Jason Gilmore
Technical Reviewer: Brian Wilson
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Associate Publisher: Grace Wong
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole LeClerc
Copy Editor: Ami Knox, Susannah Pfalzer
Assistant Production Director: Kari Brooks-Copony
Production Editor: Ellie Fountain
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: Linda Seifert
Indexer: Valerie Perry
Artist: Kinetic Publishing Services, LLC
Interior Designer: Van Winkle Design Group
Cover Designer: Kurt Krames
Manufacturing Manager: 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, e-mail 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, e-mail info@apress.com, or visit http://www.apress.com.

The sample files and source code for this book is available to readers at http://www.apress.com in the Downloads section.

CHAPTER 8

Common DNS Tasks

This chapter describes a number of common configurations when working with zone files and in some cases with BIND. These solutions are presented to assist you to quickly implement some commonly used features, to recover from errors, and to illustrate some of the more subtle uses of the DNS. The following topics are covered:

- *How to delegate a subdomain*: This configuration allows the domain name owner to pass the responsibility to a subdomain owner, which may be another party or another part of the organization, who will be entirely responsible for the zone files describing the subdomain.
- *How to delegate a virtual subdomain*: This configuration uses a single zone file to provide subdomain addressing, for instance, `www.us.example.com` or `www.uk.example.com`.
- *How to configure fail-over mail servers*: The configuration allows backup mail servers to be provided to support a domain.
- *How to reverse-map subnets*: This configuration allows the delegation of reverse mapping to subnets of typically less than 256 IPv4 addresses.
- *How to load balance with DNS*: The configurations describe various ways in which load balancing may be implemented using DNS features. The BIND statements that control the order in which addresses are returned are also covered.
- *How to define an SPF record*: The Sender Policy Framework (SPF) is an antispam measure that allows an e-mail server to verify that the SMTP source is valid for the sending e-mail address. SPF records are currently implemented by Microsoft, Google, and AOL to name but three of the many hundreds of thousands of users.
- *How to support* `http://example.com`: The configuration allows both the URL `www.example.com` and `example.com` to directly address a web or other service. The required changes to the Apache server are also covered.
- *How to fix an out-of-sequence SOA serial number*: The process used to fix various SOA serial number errors is covered.
- *How to use DNS wildcards*: The DNS RRs support the use of a wildcard (*). The section on wildcards illustrates the use of this error-prone feature.

The examples shown use a number of BIND's `named.conf` statements, which are described in Chapter 12, and standard Resource Records, which are defined in Chapter 13. If you are

running name server software other than BIND, the zone files will remain the same, but the configuration statements may differ.

In the next section, the process of delegation of a subdomain, `us.example.com`, is described to illustrate the general principle of delegation within an owner's domain name space. The domain owner can delegate everything to the *right* of the domain name in any way that makes sense—or for that matter that doesn't make sense!

Delegate a Subdomain (Subzone)

This solution configures a zone to *fully delegate* the responsibility for a subdomain to another name server. This is not the only possible method of defining subdomains—another solution involves configuring what this book calls a *virtual* or *pseudo* subdomain, which uses a single zone file to provide subdomain addressing structures. Assume the following addressing structure is required for the subdomain:

- *Zone (domain) name*: `example.com`
- *Domain host name*: `bill.example.com`
- *Subdomain name*: `us.example.com`
- *Subdomain host or service name*: `ftp.us.example.com`

To ease the zone administration load, this solution assumes the responsibility for the subdomain will be fully delegated to the `us.example.com` administrator who will be responsible for the subdomain zone files and their supporting name servers. The zone administrators of the corporate domain for `example.com` want nothing further to do with `us.example.com` other than it has generously agreed to act as the slave DNS for the subdomain name servers. When dealing with subdomains, it is important to remember that as far as the Internet registration authorities and the TLD servers are concerned, subdomains do not exist. All queries for anything that ends with `example.com` will be *referred* to the name servers for the `example.com` zone or domain. In turn, these name servers are responsible for referring the query to the subdomain name servers. For the want of any better terminology, the name servers for `example.com` are called the *domain name servers* and are visible to the gTLD `.com` servers; the name servers for `us.example.com` are called the *subdomain name servers* and are visible only to the *domain name servers* (they are invisible to the gTLD servers).

■ **Note** The term *subzone* was originally defined in RFC 1034 to describe what is today most commonly called a *subdomain*. This book uses the term *subdomain* throughout.

Domain Name Server Configuration

The following is a fragment from BIND's `named.conf` file controlling the `example.com` domain name servers:

```
// named.conf file fragment

zone "example.com" in{
                type master;
                file "master.example.com";
};
// optional - example.com acts as the slave (secondary) for the delegated subdomain
zone "us.example.com" IN {
                type slave;
                file "slave.us.example.com";
                masters {10.10.0.24;};
};
```

The optional definition of a slave (Secondary) name server for our delegated `us.example.com` subdomain is good practice but not essential. The subdomain can use any suitable name server. The zone file `master.example.com` will contain the domain configuration supporting two name servers for both the domain and the subdomain. The following zone file fragment shows this configuration:

```
; zone fragment for 'zone name' example.com
$TTL 2d     ; default TTL is 2 days
$ORIGIN example.com.
@        .  IN      SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
               12h         ; refresh =  12 hours
               15m        ; update retry = 15 minutes
               3w12h      ; expiry = 3 weeks + 12 hours
               2h20m      ; minimum = 2 hours + 20 minutes
               )
; main domain name servers
              IN      NS     ns1.example.com.
              IN      NS     ns2.example.com.
; mail domain mail servers
              IN      MX      mail.example.com.
; A records for preceding name servers
ns1           IN      A      192.168.0.3
ns2           IN      A      192.168.0.4
; A record for preceding mail server
mail          IN      A      192.168.0.5
....
; subdomain definitions in the same zone file
; $ORIGIN directive simplifies and clarifies definitions
$ORIGIN us.example.com. ; all subsequent RRs use this ORIGIN
; two name servers for the subdomain
@             IN      NS     ns3.us.example.com.
; the preceding record could have been written without the $ORIGIN as
; us.example.com. IN NS ns3.us.example.com.
```

```
; or @         IN    NS ns3
; the second name server points back to preceding ns1
                  IN    NS    ns1.example.com.
; A records for name server ns3 required - the glue record
ns3           IN      A      10.10.0.24 ; glue record
; the preceding record could have been written as
; ns3.us.example.com. A 10.10.0.24 if it's less confusing
```

The preceding fragment makes the assumptions that the domain name server `ns1.example.com` will act as a slave for the `us.example.com` subdomain. If this is not the case, any other name server can be defined the same way; but if this second name server also lies in the `us.example.com` domain, then it will require an A RR. The A RR for `ns3.example.com` for the preceding subdomain is the so-called *glue* record (see the "Glue Records in DNS" sidebar). It is necessary to allow a DNS query for the subdomain to return a referral containing both the name of the name server and its IP address. IP addresses are always defined using an A RR (or an AAAA RR if IPv6).

GLUE RECORDS IN DNS

Strictly speaking, glue records (the IP address of the name server defined using an A or AAAA RR) are only required for every name server lying *within* the domain or zone for which it is a name server. The query response—the referral—*must* provide both the name and the IP address of the name servers that lie within the domain being queried. In practice, the Top-Level Domain (TLD) servers provide the IP address for every Second-Level Domain (SLD) name server, whether in the domain or not, in order to minimize the number of query transactions. When a query to a Generic Top-Level Domain (gTLD) is issued, this name server provides the glue records for all the SLD domain's name servers. These glue records were defined and captured when the domain was registered. In the preceding configuration, the domain name server is acting in this role and must supply the IP addresses of the name servers in response to subdomain queries. To satisfy this requirement, the A RR for the name server (`ns3.us.example.com`) is a glue record and must be present. The reason a glue record *must* exist for servers *within* the domain, but is required only for performance reasons for those in a foreign or external domain, can be illustrated by looking at what would happen if the glue record were not present. If we assume the query to the gTLD server for `example.com` returned the name but not the IP address of `ns1.example.com`, then a further query would be required for the A record of `ns1.example.com`; but since the IP of the SLD name server is not yet known, it must requery the gTLD server, which answers again with the name but not the IP . . . and so on ad infinitum. Name servers for a domain (for instance, `example.com`) that lie in another domain (for instance, `ns1.example.net`) only *need* the name, since a normal query for the A RR of `ns3.example.net` will return the required IP. As noted earlier, to increase performance the IP addresses of all name servers for a domain, whether the name servers lie in the queried domain or not, are always returned by root and TLD name servers.

Subdomain Name Server Configuration

The BIND `named.conf` file controlling the subdomain name servers will contain statements similar to the following fragment:

```
// named.conf file fragment for the subdomain us.example.com

zone "us.example.com" in{
                type master;
                file "master.us.example.com";
};
```

The file `master.us.example.com` will contain the subdomain (`us.example.com`) configuration and use the two name servers that were defined in the preceding domain fragment. Here is a fragment of the subdomain zone file:

```
; zone file for subdomain us.example.com
$TTL 2d ; zone default of 2 days
$ORIGIN us.example.com.
         IN     SOA   ns3.us.example.com. hostmaster.us.example.com. (
         2003080800 ; serial number
         2h         ; refresh =  2 hours
         15m        ; update retry = 15 minutes
         3w12h      ; expiry = 3 weeks + 12 hours
         2h20m      ; minimum = 2 hours + 20 minutes
         )
; subdomain name servers
                  IN      NS     ns3.us.example.com.
                  IN      NS     ns1.example.com. ; see following notes
; subdomain mail server
                  IN      MX 10  mail.us.example.com.
; preceding record could have been written as
;                 IN      MX 10  mail
; A records for preceding name servers
ns3               IN      A      10.10.0.24
ns1.example.com.  IN      A      192.168.0.3 ; 'glue' record
; A record for preceding mail server
mail              IN      A      10.10.0.25
; next record defines our ftp server
ftp               IN      A      10.10.0.28
; the preceding record could have been written as
; ftp.us.example.com. A 10.10.0.24 if it's less confusing
....
; other subdomain records
....
```

The preceding fragment makes the assumption that `ns1.example.com` will act as a slave server for the `us.example.com` subdomain. If this is not the case, other name servers could be defined in a similar manner. The A record for `ns1.example.com` is a so-called glue record and is not strictly necessary because it is already available from a previous query. This point is worth emphasizing further since it illustrates the nature of the DNS hierarchy. To make any query for the subdomain `us.example.com`, the `example.com` domain *must have been queried first.* Since `ns1.example.com` is one of the name servers for `example.com`, its IP address is already known

to the DNS that issues the subdomain query. If the second name server for the subdomain lies in an external or foreign domain, there is no need for the glue record. Its inclusion, however, will speed up query response by removing the need for a further query to the root and gTLD servers if the external domain is not present in the cache. An external name server definition would use a standard A RR as shown in the following fragment:

```
; zone fragment for subdomain  us.example.com
;  second name server is external domain
                              IN      NS     ns1.example.net.
; A records for external name server
ns1.example.net.  IN      A      172.17.0.24
```

The FTP service host, and any others required, are only defined in the subdomain zone file and are not visible in the domain name-server zone file.

Virtual Subdomains

This solution defines what this book calls a *virtual* or *pseudo* subdomain in which the domain and the subdomain definitions appear in the same zone file. Subdomains may also be *fully delegated*, and this is the subject of a previous solution. The advantage of this configuration is that unlike a fully delegated subdomain, no additional name servers are required while still creating the subdomain style addressing structure. The disadvantage is that all changes to both the domain and the subdomain will require reloading of the main zone file. The addressing structure required is assumed to be the following:

- *Zone (domain) name*: `example.com`
- *Domain host name*: `bill.example.com`
- *Subdomain name*: `us.example.com`
- *Subdomain host name*: `ftp.us.example.com`

This solution assumes that for operational reasons the owner has decided to maintain all the information for `example.com` and `us.example.com` in a single zone file.

Domain Name Server Configuration

The BIND `named.conf` file will contain statements similar to the following fragment defining the zone `example.com` as normal:

```
// named.conf file fragment

zone "example.com" in{
                type master;
                file "master.example.com";
};
```

The file `master.example.com` will contain the domain and subdomain configuration and support two name servers.

```
; zone fragment for example.com
$TTL 2d ; zone TTL default = 2 days
$ORIGIN example.com.
@               IN      SOA   ns1.example.com. root.example.com. (
                2003080800 ; serial number
                2h         ; refresh =  2 hours
                15m        ; update retry = 15 minutes
                3w12h      ; expiry = 3 weeks + 12 hours
                2h20m      ; minimum = 2 hours + 20 minutes
                )
; main domain name servers
               IN      NS     ns1.example.com.
               IN      NS     ns2.example.com.
; mail servers for main domain
               IN      MX 10  mail.example.com.
; A records for preceding name servers
ns1            IN      A      192.168.0.3
ns2            IN      A      192.168.0.4
; A record for preceding mail servers
mail           IN      A      192.168.0.5
; other domain-level hosts and services
bill           IN      A      192.168.0.6
....
; subdomain definitions
$ORIGIN us.example.com.
               IN      MX 10  mail
; preceding record could have been written as
; us.example.com.   IN  MX 10 mail.us.example.com.
; A record for subdomain mail server
mail           IN      A      10.10.0.28
; the preceding record could have been written as
; mail.us.example.com. A 10.10.0.28 if it's less confusing
ftp            IN      A      10.10.0.29
; the preceding record could have been written as
; ftp.us.example.com. A 10.10.0.29 if it's less confusing
....
; other subdomain definitions as required
$ORIGIN uk.example.com.
....
```

Additional subdomains could be defined in the same file using the same strategy. For administrative convenience, the standard zone file `$INCLUDE` directive may be used to include the subdomain RRs as demonstrated in the following fragment:

```
; fragment from zone file showing use of $INCLUDE
....
; other domain-level hosts and services
bill          IN      A      192.168.0.5
....
; subdomain definitions
$INCLUDE sub.us.example.com
; other subdomain definitions as required
```

This solution illustrates that subdomain addressing can be easily accomplished in a single zone file at the possible cost of administrative convenience. This structure, as well as being simpler than a *fully delegated* subdomain, does not require any additional name servers.

Configure Mail Servers Fail-Over

This solution is provided for the sake of completeness and uses material already covered in Chapter 2. It configures a DNS server to provide fail-over or alternate mail service when the primary mail service is off-line or not accessible for a period of time. It involves use of the *preference* field of the MX RRs (see Chapter 13) as shown in the following fragment:

```
; zone file fragment
      IN  MX  10 mail.example.com.
      IN  MX  20 mail.example.net.
....
mail  IN  A      192.168.0.4
....
```

If the most preferred mail server, the one with the lowest number (10), which in the preceding fragment is `mail.example.com`, is not available, mail will be sent to the second most preferred server, the one with the next highest number (20), which in the preceding fragment is `mail.example.net`. The secondary mail server (`mail.example.net`), which would ideally be located at a separate geographic location, would typically be configured as a simple relay (or forwarding) mail server with a very long retry time, in which case it will accept the mail and try and relay it to the proper destination (`mail.example.com`) over the next six weeks or whatever you configure the retry time to be.

Delegate Reverse Subnet Maps

This solution describes how to delegate reverse mapping for subnets. Delegation of reverse subnet maps may be used by ISPs or other service providers as a means to enable a user of a static IP range, delegated from the service provider, to be responsible for their own reverse-mapping zone files. In the example shown, a subnet is defined to be less than 256 IPv4 addresses though the solution could be used for any part of an IPv4 address range. Normal reverse mapping is described in Chapter 3 and in our example case is assumed to reverse map down to the third element of an IPv4 address; for instance, assume an IPv4 address of 192.168.199.15, then normal reverse mapping will typically cover the 192.168.199 part, which is then reversed and placed under the domain `IN-ADDR.ARPA`, giving `199.168.192.IN-ADDR.ARPA`. The resulting reverse map

will contain the hosts from 192.168.199.0 to 192.168.199.255. We now assume that subnets of 64 addresses are assigned to four separate users (192.168.199.0/26, 192.168.199.64/26, 192.168.199.128/26, 192.168.199.192/26) and to minimize work the assignee wishes to delegate responsibility for reverse mapping to the subnet users (the assignors). The reverse map has been delegated once to the assignee of 192.168.199.0 and cannot therefore be delegated again. Our assignee must use a special technique defined in RFC 2317. The technique involves creating additional space in the reverse-map address hierarchy. Both the assignee and the assignor (end user) are required to implement the technique in their zone files; examples of both are shown in the upcoming text.

Assignee Zone File

The following fragment shows the 192.168.199.64/26 subnet as a fragment of a reverse-map zone file located at the assignee (using the `example.net` domain) of the subnet:

```
; zone file fragment for example.net
$TTL 2d ; zone default TTL = 2 days
$ORIGIN 199.168.192.IN-ADDR.ARPA.
@            IN  SOA   ns1.example.net. hostmaster.example.net. (
                              2003080800 ; serial number
                              2h         ; refresh
                              15m        ; update retry
                              2w         ; expiry
                              3h         ; minimum
                              )
              IN  NS      ns1.example.net.
              IN  NS      ns2.example.net.
; definition of other IP address 0 - 63
....
; definition of our target 192.168.199.64/26 subnet
; name servers for subnet reverse map
64/26         IN  NS  ns1.example.com.
64/26         IN  NS  ns2.example.com.
; the preceding could have been written as
; 64/26.199.168.192.IN-ARDDR.ARPA.  IN NS ns2.example.com.
; IPs addresses in the subnet - all need to be defined
; except 64 and 127 since they are the subnets multicast
; and broadcast addresses not hosts/nodes
65            IN  CNAME   65.64/26.199.168.192.IN_ADDR.ARPA. ;qualified
66            IN  CNAME   66.64/26 ;unqualified name
67            IN  CNAME   67.64/26
....
125           IN  CNAME   125.64/26
126           IN  CNAME   126.64/26
; end of 192.168.199.64/26 subnet
.....
; other subnet definitions
```

The method works by forcing the CNAME lookup to use the name servers defined for the subnet; that is, the address 65 will find the `CNAME 65.64/26.199.168.192.IN-ADDR.ARPA.`, which is *resolved* to the name servers `ns1.example.com.` and `ns2.example.com.`, both of which in this case are located at the assignor (end user). The `64/26` name, which makes the additional name space look like a IP prefix or slash notation address, is an artificial, but legitimate, way of constructing the additional space to allow delegation. The `/` (slash) relies on a liberal interpretation of the rules for a name or *label* (RFC 2181), but it could be replaced with `-` (dash)—for instance, `64-26`, if that makes you more comfortable. Any number of subnets of variable size can be assigned in this manner; that is, the subnet following the one defined previously could be `128/27` (32 IP addresses) or `128/28` (16 addresses) or `128/25` (128 IP addresses). No changes are required to the BIND configuration to support this reverse map.

Assignor (End-user) Zone File

The zone file for the reverse map (`ns1.example.com` in this example) is a conventional reverse map and looks like this:

```
$TTL 2d ; zone default = 2 days
$ORIGIN 64/26.199.168.192.IN-ADDR.ARPA.
@            IN  SOA   ns1.example.com. hostmaster.example.com. (
                                2003080800 ; serial number
                                2h         ; refresh
                                15m        ; update retry
                                2w         ; expiry
                                3h         ; minimum
                                )
              IN  NS      ns1.example.com.
              IN  NS      ns2.example.com.
; IPs addresses in the subnet - all need to be defined
; except 64 and 127 since they are the subnets multicast
; and broadcast addresses not hosts/nodes
65            IN  PTR   fred.example.com. ;qualified
66            IN  PTR   joe.example.com.
67            IN  PTR   bill.example.com.
....
125           IN  PTR   web.example.com.
126           IN  PTR   ftp.example.com.
; end of 192.168.23.64/26 subnet
```

Finally, the reverse-map `zone` clause in the `named.conf` file needs to be changed to reflect the revised zone name. The following example shows the reverse-map zone clause fragment:

```
// named.conf fragment at example.com
// revised reverse-map zone name
zone "64/26.199.168.192.IN-ADDR.ARPA" in{
                type master;
                file "192.168.23.rev";
};
```

■ **Note** The technique used in the preceding method is credited to Glen A. Herrmannsfeldt, who is obviously a very creative person and one might conjecture had problems persuading his ISP to delegate reverse-mapping responsibility.

DNS Load Balancing

These solutions use the DNS to configure various forms of load balancing. In this context load balancing is defined as the ability to use standard DNS services to share the load between two or more servers providing the same or similar services. The section covers the following topics:

- Balancing mail
- Balancing other services (for instance, web or FTP)
- Balancing services using the SRV RR
- Controlling the order of RRs

This section ends with a brief discussion of the effectiveness of DNS-based load-balancing strategies.

Balancing Mail

Mail is unique in that two possible strategies may be used. The following fragment shows use of multiple MX records with equal-preference values:

```
; zone file fragment
        IN  MX  10  mail.example.com.
        IN  MX  10  mail1.example.com.
        IN  MX  10  mail2.example.com.
....
mail    IN  A       192.168.0.4
mail1   IN  A       192.168.0.5
mail2   IN  A       192.168.0.6
```

The name sever will deliver the MX RRs in the order defined by the `rrset-order` statement (defined later in this section and fully in Chapter 12) and which defaults to round-robin (or cyclic) order. The requesting SMTP server will then apply its algorithm to select one from the equal-preference list that *may* work *against* the BIND `rrset-order` statement. Currently sendmail (8.3.13), Exim (4.44), and Postfix (2.1) all have documented references to indicate they use a random algorithm for records of equal preference; indeed, Postfix allows control over the behavior using the `smtp_randomize_addresses` parameter (default is yes). In this case, the randomizing algorithm may select the very IP that BIND's `rrset-order` algorithm positioned, say, last in the returned order. Documentation for qmail, courier-mta, and Microsoft (Exchange and IIS SMTP) does not describe what these packages do with equal-preference MX values. An alternative approach is to use multiple A records with the same name and different IP addresses as shown in this fragment:

```
; zone file fragment
          IN  MX  10  mail.example.com.
....
mail      IN  A       192.168.0.4
          IN  A       192.168.0.5
          IN  A       192.168.0.6
```

The name server will deliver the A RRs in the order defined by the `rrset-order` statement in BIND's `named.conf` file. In order to satisfy reverse lookup requests used by most mail servers for simple authentication, all the IP addresses listed must be reverse mapped to `mail.example.com` as shown in the following fragment:

```
; reverse-map file fragment
; for 0.168.192.IN-ADDR.ARPA
....
4           PTR      mail.example.com.
5           PTR      mail.example.com.
6           PTR      mail.example.com.
```

The net effect of the two methods is the same. In the case of equal-preference MX records, the control of load lies with the SMTP server's algorithm. In the case of multiple A RRs, control lies with the name server, which in the case of BIND provides the `rrset-order` statement to select the order of A RRs (RRsets) as well as other RRsets. In both the preceding cases, each mail server must be capable of either synchronizing mailbox delivery or all but one of the servers must be mail relays or forwarders.

Balancing Other Services

This section illustrates load balancing with web and FTP services, but the same principle applies to any service. In this case the load-balancing solution uses multiple A RRs as shown in the following fragment:

```
; example.com zone file fragment
....
ftp   IN  A   192.168.0.4
ftp   IN  A   192.168.0.5
ftp   IN  A   192.168.0.6
www   IN  A   192.168.0.7
www   IN  A   192.168.0.8
```

This RR format, which relies on blank name replication, produces exactly the same result:

```
; example.com zone file fragment
....

ftp   IN  A   192.168.0.4
      IN  A   192.168.0.5
      IN  A   192.168.0.6
www   IN  A   192.168.0.7
      IN  A   192.168.0.8
```

The name server will deliver all the IP addresses defined for the given name in answer to a query for the A RRs; the order of IP addresses in the returned list is defined by the `rrset-order` statement in BIND's `named.conf` file. The FTP and web servers must all be exact replicas of each other in this scenario.

Balancing Services

The SRV record provides load balancing by using both a *priority* field and a *weight* field for fine-grained control as well as providing fail-over capability. The SRV RR description in Chapter 13 contains an example illustrating its use in load balancing. The SRV RR is not yet widely supported at this time with two notable exceptions: Lightweight Directory Access Protocol (LDAP), which was partly responsible for development for the SRV record and which is used as a part of the discovery process for LDAP servers and the Session Initiation Protocol (SIP) used in VoIP.

Controlling the RRset Order

BIND versions after 9.2.3 fully implement the `rrset-order` statement, which can be used to control the order in which equal RRs, an RRset, of *any* type are returned. The `rrset-order` statement can take a number of arguments, which are described in Chapter 12, but the following fragment only uses the `order` keyword, which may take the values `fixed`—the order the records were defined in the zone file; `cyclic`—start with the order defined in the zone file and round-robin for each subsequent query; and `random`—randomly order the responses for every query. The `rrset-order` statement can only appear in the global `options` clause for BIND but can take addition arguments that can make it applicable to one or more zones. An example `named.conf` fragment follows that will return any RRset (a set of equal RRs) in round-robin order:

```
// named.conf fragment
options {
// other options
            rrset-order {order cyclic;};
};
```

Assume a zone file has the following MX records:

```
; zone file fragment for example.com
             MX  10   mail1.example.com.
             MX  10   mail2.example.com.
             MX  10   mail3.example.com.
```

The first query to this zone for MX records will return in the order `mail1.example.com.`, `mail2.example.com.`, `mail3.example.com.`; the second query will return `mail3.example.com.`, `mail1.example.com.`, `mail2.example.com.`; and so on in cyclic (or round-robin) order.

Effectiveness of DNS Load Balancing

Clearly the effects of caching can significantly distort the effectiveness of any DNS IP address allocation algorithm. A TTL value of 0 may be used to inhibit caching or the increasingly common very short TTL values (30–60 seconds) could be used to reduce the potentially negative caching effect, but only at the cost of a significant rise in the number of DNS queries. It would

be a little unfortunate to achieve excellent load balancing across two or three web servers at the cost of requiring ten more name servers. Intuition, without serious experimentation, would suggest that assuming a normal TTL (12 hours or more) and *any* changing IP allocation algorithm (cyclic or random) loads would be reasonably balanced (measured by request arrivals at an IP) given the following assumptions:

- Traffic is balanced over a number of DNS caches, that is, traffic originates from a number of ISPs or customer locations where DNS caches are maintained. Specifically there are no *pathological* patterns where 90% (or some largish number) of the load originates from one particular cache.
- The volume of traffic is reasonably high since pathological patterns are more likely in small traffic volumes.

DNS load balancing cannot, however, account for service loading; for instance, certain transactions may generate very high CPU or resource loads. For this type of control only a specialized load balancer—which measures transaction response times from each server—will be effective.

Define an SPF Record

This section defines how to configure a Sender Policy Framework (SPF) record for a domain and its mail servers. SPF is being proposed as an IETF experimental standard to enable validation of legitimate sources of e-mail. The information found in this section is based on the current versions of the SPF specification and may change—current information may be obtained from `www.ietf.org/internet-drafts/draft-schlitt-spf-classic-01.txt`.

The design intent of the SPF record is to allow a receiving *Message Transfer Agent (MTA)* to verify that the originating IP (the *source-ip*) of an e-mail from a *sender* is authorized to send mail for the *sender*'s domain. The SPF information is contained in a standard TXT RR (though a new RR type may be allocated if and when SPF reaches standardization by the IETF). The TXT RR is described in Chapter 13. If an SPF (TXT) RR exists and authorizes the source IP address, the mail can be accepted by the MTA. If the SPF (TXT) RR does not authorize the IP address, the mail can be bounced—it did not originate from an authorized source for the *sender*'s domain. If the domain does not have an SPF RR, the situation is no worse than before. Many commercial and Open Source MTAs have already been modified to use the SPF record, including sendmail, qmail, Postfix, courier, Exim, and Microsoft Exchange to name but a few. There is a less widely implemented proposal from Yahoo! called DomainKeys (`http://antispam.yahoo.com/domainkeys`), which is a cryptographic-based solution. Microsoft is advocating a standard called Send ID,[1] which contains SPF as a subset but adds a new Purported Responsible Address (PRA) field to the e-mail to provide additional checking.

This solution both describes the format of the SPF record and presents a number of example configurations. The following terminology is used to simplify the subsequent descriptions:

- *Sender*: The full e-mail address of the originator of the mail item (obtained from the *return path* in the actual SPF checks), for instance, `info@example.com`.

[1] More information on Send ID can be found at `www.microsoft.com/mscorp/twc/privacy/spam/senderid/overview.mspx`.

- *Sender-ip*: The IP address of the SMTP server trying to send this message, for instance, 192.168.0.2.
- *Sender-domain*: The domain name part of the *sender*'s e-mail address, for instance, assume the *sender* is `info@example.com`, then the *sender-domain* is `example.com`.

The SPF record defines one or more tests to verify the *sender*. Each test returns a condition code (defined by the `pre` field shown in the next section). The first test to pass will terminate SPF processing.

TXT RR Format

The standard TXT RR format is defined as follows:

```
name  ttl  class   rr     text
```

The SPF record is entirely contained in the `text` field (a quoted string). SPF defines the contents of the quoted string as shown here:

```
"v=spf1 [pre] type [[pre] type] ... [mod]"
```

SPF records are normally defined for the domain and the mail server(s). The following shows a zone file fragment containing SPF records for the domain and the mail server, which in this case only allows mail for the domain to be sent from the host `mail.example.com`.

```
; zone file fragment for example.com
              IN  MX 10 mail.example.com.
....
mail          IN  A     192.168.0.4
; SPF records
; domain SPF
example.com.  IN  TXT   "v=spf1 mx -all"
; mail host SPF
mail          IN  TXT   "v=spf1 a -all"
```

The following text describes the fields used in an SPF record and references where appropriate the `v=spf1 mx -all` SPF record from the preceding example fragment:

`v=spf1` Field

This field is mandatory and defines the version being used. Currently the only version supported is spf1.

`pre` Field

This optional (defaults to +) field defines the code to return when a match occurs. The possible values are + = pass (default), - = fail, ~ = softfail (indeterminate result), ? = neutral. If a test is conclusive, either add + or omit (defaults to +) as in the first test in the example fragment, which could have been written as `+mx`. If a test might not be conclusive, use ? or ~. - is typically only used with `-all` to indicate the action if there have been no previous matches as in the terminating test from the same fragment.

type Field

This defines the mechanism type to use for verification of the *sender*. Multiple type tests may be defined in a single SPF record. In the example fragment, there are two type tests, mx (or +mx) and all (-all). Each of the type values is described in detail in the section "SPF Type Values."

mod Field

Two optional record modifiers are defined. If present, they should follow the last type directive, that is, after the terminating all. The current values defined are as follows:

redirect=domain Field

This redirects verification to use the SPF record of the defined domain. This format may be used to enable additional processing in the event of a failure or may be used on its own in an SPF to provide a single domain-wide definition. This format is the same as the type include but may be used without the terminating all type.

This SPF allows additional processing using the SPF for example.net if the mail from example.com tests fail:

```
        IN  TXT :"v=sfp1 mx ?all redirect=example.net"
```

This SPF redirects all processing for example.com to a standard SPF record in the domain example.net:

```
       IN  TXT "v=spf1 redirect=_spf.example.net"
```

The zone file for example.net would include the following record:

```
_spf      IN  TXT  "v=spf1 mx -all"
```

exp=text-rr Field

The exp type, if present, should come last in an SPF record (after the all type if present). It defines the name of a TXT record, text-rr, whose text may be optionally returned with any failure message. This fragment shows a trivial example where the *sender* of the mail is informed that they are not authorized to send mail. More complex examples, including the use of macro expansion, can be constructed, referring users to a site that could inform them of the procedure to define SPF records.

```
; domain example.com SPF record
        IN  TXT "v=spf1 mx -all exp=getlost.example.com"
; the getlost TXT record
getlost  IN  TXT "You are not authorized to send mail for the domain"
```

The text field is allowed to contain macro expansions as described in the section "Macro Expansion."

SPF type Values

The SPF type parameter defines either the mechanism to be used to verify the *sender* or to modify the verification sequence as described in the following sections.

Basic Mechanisms

These types do not define a verification mechanism but affect the verification sequence:

- `include:domain`: Recurse testing using the supplied domain. The SPF record for `domain` replaces the *sender-domain*'s SPF and processing uses the rules defined in the `included` SPF. This is the most common form when clients send mail through an ISP's servers.
- `all`: Terminates a test sequence if no positive results have been found previously.

Sender Mechanisms

These types define a verification mechanism.

Type `ip4` Format

This type may take one of the following formats:

```
ip4:ipv4 ip4:ipv4/cidr
```

The `ip4` type uses the *sender-ip* for verification. If the *sender-ip* is the same as `ipv4`, the test passes. This may take the additional argument `ip4:ipv4/cidr`, in which case if the source IPv4 address lies in the range defined by `cidr`(the IP prefix or slash notation), the test passes. This type uses no additional DNS resources and is therefore the recommended solution for IPv4.

This SPF only allows e-mail for the domain to be sent from `192.168.0.2`:

```
IN  TXT "v=spf1 ip4:192.168.0.2 -all"
```

This SPF allows mail to be sent from any of the 32 addresses that contains the address 192.168.0.38 (CIDR range is from 192.168.0.32–63):

```
IN  TXT "v=spf1 ip4:192.168.0.38/27 -all"
```

Type `ip6` Format

This type may take one of the following formats:

```
ip6:ipv6 ip6:ipv6/cidr
```

The `ip6` type uses the same formats defined for `ip4` previously. This type uses no additional DNS resources and is therefore the recommended solution for IPv6.

The following only allows messages for the domain to be sent from the single address 2001:db8:0:0:0:0:0:10:

```
IN  TXT "v=spf1 ip6:2001:db8::10 -all"
```

The next example allows mail to be sent from 32 addresses that contain the address 2001:db8:0:0:0:0:0:10 (range is from 2001:db8:0:0:0:0:0:1 to 2001:db8:0:0:0:0:0:1f).

```
IN  TXT "v=spf1 ip4:2001:db8::10/123 -all"
```

Type `a` Format

This type may take one of the following formats:

```
a a/cidr a:domain a:domain/cidr
```

The a type uses an A RR for verification. In the basic format with no additional arguments, if the A RR for the *sender-domain* is the same as the *sender-ip*, the test passes. The optional form a/cidr will apply the test to the extended range defined by the IP prefix (or slash) notation. The form a:domain will cause the test to be applied to the A RR of domain, and a:domain/cidr will apply the test to the range of IPs defined by the IP prefix (or slash) notation. The domain argument may also use macro expansion, defined later in this section. The a and a/cidr formats require an A RR for the domain as shown in the following fragment:

```
; zone fragment for example.com
$ORIGIN example.com.
...
@             IN   A       192.168.0.2
              IN   TXT "v=spf1 a -all"
....
```

This SPF allows only the host smtp.example.net to send mail for the domain example.com:

```
              IN   TXT "v=spf1 +a:smtp.example.net -all"
```

The advantage of using the preceding construct is that if the IP address of smtp.example.com changes, the preceding SPF record does not change. The cost, however, is one more DNS transaction for every SPF check.

Type mx Format

This type may take one of the following formats:

```
mx mx/cidr mx:domain mx:domain/cidr
```

The mx type uses the MX RRs and the mail server A RRs for verification. Remember, this type uses the MX RR for the domain, which may not be the same as the SMTP server for the domain. In the basic format with no additional arguments, the MX record for the *sender-domain* and the A RRs for the defined mail host(s) are obtained; if the IP address of the *sender-ip* matches any of the mail host IPs, the test passes. The form mx:/cidr applies the address range defined by cidr (IP prefix or slash notation) is used for the match. The format mx:domain uses the MX and A RRs for domain instead of the *sender-domain*, and the format mx:domain/cidr extends the IP address check to the cidr (IP prefix or slash notation) range of IP addresses. The domain argument may also use macro expansion defined later in this section. Use of the mx format involves at least two DNS lookups per SPF verification operation.

This SPF allows mail from the domain example.com to be sent from any mail server defined in an MX RR for the domain example.net:

```
              IN  TXT "v=spf1 mx:example.net -all"
```

This SPF allows mail to be sent from any of the 16 IP addresses containing each of the mail servers defined in MX records for the sending domain:

```
              IN  TXT "v=spf1 mx/28-all"
```

Type ptr Format

This type may take one of the following formats:

```
ptr ptr:domain
```

The ptr type uses PTR RRs of the *sender-ip* for verification. In the basic format with no additional arguments, the *sender-ip* is used to query for the host name using the reverse map. The A or AAAA RR for the resulting host is then obtained. If this IP matches the *sender-ip and* the *sender-domain* is the same as the domain name of the host obtained from the PTR RR, then the test passes. The form ptr:domain replaces the *sender-domain* with domain in the final check for a valid domain name. The domain argument may also use macro expansion defined later in this section. The PTR record is the least preferred solution since it places a load on the IN-ADDR.ARPA (IPv4) or IP6.ARPA (IPv6) reverse-map domains, which generally have less capacity than the gTLD and ccTLD domains.

This SPF would allow any host in the domain example.com that is reversed mapped to send mail for the domain:

```
        IN  TXT "v=spf1 ptr -all"
```

Type exists Format

This type may take one of the following formats:

```
exists exists:domain
```

The exists type tests for existence of the *sender-domain* using an A RR query. In the basic format with no arguments, an A RR query is issued using the *sender-domain* and if any result is obtained, the test passes. The form exists:domain applies the same test but for domain. The domain argument may also use macro expansion defined later in this section. The exists form requires an A RR for the domain as shown in the following zone file fragment:

```
; zone fragment for example.com
$ORIGIN example.com.
...
@             IN   A       192.168.0.2
              IN   TXT "v=spf1 +exists -all"
....
```

Macro Expansion

The SPF record allows macro expansion features using a %(x) format where % indicates a macro and x is a character defining the macro type as defined in Table 8-1.

Table 8-1. *SPF Macro Expansion Arguments*

Macro	Function
%(c)	Only allowed in TXT records referenced by the exp field. The IP of the receiving MTA.
%(d)	The current domain, normally the *sender-domain* %(o), but replaced by the value of any domain argument in the type field as discussed previously.
%(h)	The domain name supplied on HELO or EHLO, normally the host name of the sending SMTP server.
%(i)	The *sender-ip* value. The IP of the SMTP server sending mail for user info@example.com.
%(l)	Replace with local part of *sender*. For instance, if the *sender* is infor@example.com, the local part is info.
%(o)	The *sender-domain* value. For instance, if the e-mail address is info@example.com, the *sender-domain* is example.com.
%(p)	The validated domain name. The name obtained using the PTR RR of the *sender-ip*. Use of this macro will require an additional query unless a ptr type is used.
%(r)	Only allowed in TXT records referenced by the exp field. The name of the host performing the SPF check. Normally the same as the receiving MTA.
%(t)	Only allowed in TXT records referenced by the exp field. Defines the current timestamp.
%(s)	Replace with *sender* e-mail address, for instance, info@example.com.
%(v)	Replaced with in-addr if *sender-ip* is an IPv4 address and ip6 if an IPv6 address. Used to construct reverse-map strings.

The preceding macros may take one or more additional arguments as follows:

- r: Indicates reverse the order of the field. For instance, %(or) would display example.com as com.example, and %(ir) would display 192.168.0.2 as 2.0.168.192. The default splitting point for reversing the order uses . (dot) as the separator but any other separator may be used; for instance, %(sr@) would split info@example.com at the @ separator and when reversed will display example.com.info (when fields are rejoined they will always use a . [dot]).
- *Digit*: The presence of a digit (range 1 to 128) controls the number of rightmost names or labels displayed. For instance, %(d1) uses the d part to extract the current domain (assume its example.com) as defined previously, and the qualifying digit (1) displays only one rightmost label from the name, in this case com; but %(d5) would display five right-hand names or labels up to the maximum available, which in this example would display example.com.

SPF Record Examples

The following examples are designed to illustrate various uses of the SPF record. The SPF macro expansion features in particular can lead to complex definitions; further examples may be discovered by interrogating such domains as microsoft.com, aol.com, and google.com, all of whom are among the 750,000+ domains that currently publish SPF records. A dig command (introduced in Chapter 9) such as shown here will yield an SPF record if published:

```
# dig example.com txt
```

Substitute your favorite domain in the preceding example to verify the existence of an SPF record.

Single Domain Mail Server

This example assumes a single mail server that both sends and receives mail for the domain:

```
; zone file fragment for example.com
$ORIGIN example.com.
          IN  MX 10 mail.example.com.
....
mail      IN  A     192.168.0.4
; SPF records
; domain SPF
@         IN  TXT   "v=spf1 mx -all"
; mail host SPF
mail      IN  TXT   "v=spf1 a -all"
```

The domain SPF is returned from a *sender-domain* query using the *sender* e-mail address; for instance, the *sender* is `info@example.com`, and the *sender-domain* is `example.com`. The SPF record only allows the MX host(s) to send for the domain. The mail host SPF is present *in case* the receiving MTA uses a reverse query to obtain the *sender-ip* host name and then does a query for the SPF record of that host. The SPF record states that the A record of `mail.example.com` is permitted to send mail for the domain. If the domain contains multiple MX servers, the domain SPF would stay the same, but each mail host should have an SPF record.

SMTP Server Offsite

This example assumes the domain `example.com` will send mail through an off-site mail server in `example.net`, for instance, an ISP:

```
; zone file fragment for example.com
$ORIGIN example.com.
             IN  MX 10  mail.example.net.
....
; SPF records
; domain SPF
@            IN  TXT    "v=spf1 include:example.net -all"
; WARNING: example.net MUST have a valid SPF definition
```

This format should be used if and only if it is known that `example.net` has a valid SPF record. The `include` recurses (restarts) verification using the SPF records for `example.net`. Mail configuration changes are localized at `example.net`, which may simplify administration. The `include` could have been replaced with redirect as shown here:

```
@            IN  TXT    "v=spf1 redirect=example.com"
```

Virtual Mail Host

This example assumes `example.net` is the host for a large number of virtual mail domains and supplies SMTP services for others. The zone file fragment that follows describes one of the virtual mail domains `example.org`:

```
; zone file fragment for example.org
$ORIGIN example.org.
            IN  MX 10 mail.example.net.
....
; SPF records
; domain SPF
@          IN  TXT   "v=spf1 include:example.net -all"
```

The domain SPF is returned from a *sender-domain* query using the *sender* e-mail address; for instance, the *sender* is `info@example.org`, and the *sender-domain* is `example.org`. The SPF record recurses to the domain name `example.net` for verification.

Here is the zone file for `example.net`:

```
; zone file fragment for example.net
$ORIGIN example.net.
             IN  MX 10   mail.example.net.
....
mail         IN  A       192.168.0.37
; SPF records
; domain SPF - any host from
; 192.168.0.32 to 192.168.0.63 can send mail
; and any MX host
@            IN  TXT    "v=spf1 ip4:192.168.0.37/27 mx -all"
; mail SPF
mail         IN  TXT    "v=spf1 a -all"
```

The domain SPF is returned from a *sender-domain* query using the *sender* e-mail address; for instance, the *sender* is `info@example.net`, and the *sender-domain* is `example.net` or the `include:example.net` if the mail originated from the `example.org` zone. The SPF record allows any host in the 32 address subnet that contains 192.168.0.37 to send mail for this domain (`example.net`) and any hosted virtual domain, that is, `example.org` in the preceding example. The SPF also allows any host defined in an MX RR as an alternative if the first test fails and allows for a future reconfiguration of the network that may move the host `mail.example.net` IP address outside the defined `ip4` range. The scenario could have used a slightly shorter version:

```
@         IN  TXT    "v=spf1 mx/27 -all"
```

This record has the same effect as `a:192.168.0.37/27` but will cost a further DNS lookup operation, whereas the IP is already available. The scenario relies on the fact that customers will only send mail via the domain `example.net`, that is, they will *not* send mail via another ISP when at home or when traveling. If you are not sure if this is the case, the sequence can be terminated with `?all`, which indicates that the results may not be definite—it allows the mail to pass, perhaps after logging the incident to capture statistics. If the domain contains multiple MX servers, the domain SPF would stay the same, but each mail host would have an SPF record.

No Mail Domain

This example assumes that the domain `example.org` *never* sends mail from any location—ever. Typically this would be done to prevent bogus mail using this domain for everyone else—it is a supreme act of self-sacrifice!

```
; zone file fragment for example.org
; zone does NOT contain MX record(s)
$ORIGIN example.org.
...
; SPF records
; domain SPF
@        IN  TXT   "v=spf1 -all"
```

This SPF test will always fail since the only condition it tests is the `-all`, which, because of the - (minus), results in a fail.

Using Macro Expansion

This example uses macro expansion in the SPF and the polite message sent to users to indicate that the *sender* may be being impersonated. The zone file fragment is as follows:

```
; zone file fragment for example.com
$ORIGIN example.com.
            IN  MX 10 mail.example.com.
....
; SPF records
; domain SPF
@          IN  TXT   "v=spf1 exists:%(d) -all ext=badguy.example.com"
badguy     IN   TXT  "The email from %(s) using SMTP server at %(i) was rejected \
   by %(c) (%(r)) at %(t) \
   because it failed the SPF records check for the domain %(p). \
   Please visit http://abuse.example.com/badguys.html for more information"
```

The `badguy` TXT RR is split across multiple lines (each ending with a `\`) for presentation reasons only and should appear on a single line in the zone file. The `exists:%(d)` tests for the existence of the *sender-domain*, which is the default value for the `exists` test but is used to illustrate use of macros in expressions.

Supporting `http://example.com`

This solution configures a name server to allow URLs of the form `http://www.example.com` and `http://example.com`—both URLs will address (or resolve to) the same web server. Seems it's the cool thing to do these days. To make this feature work also requires a change to the web server. The required change to Apache when using virtual hosts is also provided.

```
; zone fragment for example.com
$TTL 2d ; zone ttl default = 2 days
$ORIGIN example.com.
....
; SOA NS MX and other records
```

```
; define an IP that will resolve example.com
@               IN      A       192.168.0.3
; you could also write the preceding line as
; example.com.   IN      A       192.168.0.3
www             IN      CNAME   example.com. ; dot essential
; aliases www.example.com to example.com
; OR define another A record for www using same host
; this is the least number of changes and saves a CNAME
www             IN      A       192.168.0.3
```

The preceding will also work for any other host name as long as different ports are in use; for instance, `ftp://example.com` will work if the FTP server was appropriately configured and on the same host, which in the preceding case is 192.168.0.3.

Apache Configuration

This configuration assumes the use of virtual hosts on an Apache (1.3.x or 2.x) server. Apache's `httpd.conf` configuration file containing the `VirtualHost` section for `example.com` would look something like the following fragment:

```
<VirtualHost 10.10.0.23>
    ServerAdmin webmaster@example.com
    DocumentRoot /path/to/web/root
    ServerName www.example.com
    ErrorLog logs/www.example.err
    CustomLog logs/www.example.log common
</VirtualHost>
```

A second `VirtualHost` definition is added with `ServerName` modified to reflect the `example.com` change as follows:

```
<VirtualHost 10.10.0.23>
    ServerAdmin webmaster@example.com
    DocumentRoot /path/to/web/root
    ServerName example.com
    ErrorLog logs/example.err
    CustomLog logs/example.log common
</VirtualHost>
```

In the preceding example, a second log and error file is used to avoid possible corruption. An alternate method is to use a single `VirtualHost` definition with the `ServerAlias` directive as shown here and which only requires single log and error files:

```
<VirtualHost 10.10.0.23>
    ServerAdmin webmaster@example.com
    DocumentRoot /path/to/web/root
    ServerName www.example.com
    ServerAlias example.com
    ErrorLog logs/example.err
    CustomLog logs/example.log common
</VirtualHost>
```

In many cases, when `example.com` is entered, your ever-helpful browser will auto-complete (or guess) that what you really meant was `www.example.com` and add the `www` automatically. So after all that hard work in many browsers, `example.com` would have worked even if you had done nothing!

■ **Caution** If you are using MS FrontPage extensions with a single `VirtualHost` definition, then the `ServerName` must be the name that is used to log in to FP. In the preceding example, the FrontPage login name used would be `www.example.com`.

Out-of-Sequence Serial Numbers

The `serial number` field of the SOA RR (described in Chapter 2) by convention uses a date format defined to be yyyymmddss where yyyy is the four-digit year number, `mm` is the two-digit month number, `dd` is the two-digit day within month number, and `ss` is a two-digit sequence number within the day. Since this is only a convention, BIND and most other DNS software does not validate the format of this field; it is very easy to introduce errors into this number and get out of sequence. Zone transfer to zone slave will, in the event of zone file changes, occur only if the serial number of the SOA RR is greater that the previous one. So the dreaded day has come and while pondering the meaning of life during a zone file update the serial number is changed, BIND has been restarted, and only with something approaching shock and awe you discover the SOA serial number is incorrect. Apart from ritual suicide, what can be done?

To illustrate the fixes possible, it is assumed that today's date is 28 February 2003 (serial number `2003022800`). If the erroneous serial number entered is less than today, that is, `2003022700`, the fix is trivial: simply correct the serial number and restart or reload BIND or reload the zone with `rndc` (see Chapter 9). If the number is too high, it depends on how high the number is and how frequently the zone file is changed. Assume the changed serial number was set to `2004022900`, which as we all know does not exist, 2003 not being a leap year; however, BIND does not know that and a zone transfer will have taken place, 29 being greater than 28. The simple fix is to increment the date again to `2003030100` and keep using the sequence number until the correct date is reached (tomorrow in this case). This works unless you will require to make more than 99 changes until the erroneous date is reached.

If all the quick solutions are not acceptable, for instance, the serial number is `2008022800`, then it's time to get out the calculator or do some serious mental arithmetic. The SOA serial number is an unsigned 32-bit field with a maximum value of $2^{31,}$ which gives a range of 0 to 4294967295, but the maximum increment to such a number is 2^{31-1} or 2147483647, incrementing the number by the maximum would give the same number. Using the maximum increment, the serial number fix is a two-step process. First, add 2147483647 to the erroneous value, for example, 2008022800 + 2147483647 = 4155506447, restart BIND or reload the zone, and make *absolutely* sure the zone has transferred to all the slave servers. Second, set the SOA serial number for the zone to the correct value and restart BIND or reload the zone again. The zone will transfer to the slave because the serial number has wrapped through zero and is

greater that the previous value of 4155506447! RFC 1982 contains all the gruesome details of serial number comparison algorithms if you are curious about such things.

Use of Wildcards in Zone Files

The standard wildcard character * (asterisk) can be used as a name with any RR. Wildcards can have unintended consequences and should only be used with considerable caution.

■ **Note** The Internet Architecture Board (IAB) has even published a paper on the subject of wildcard usage after the infamous use of wildcard A RRs by a gTLD operator to redirect users to a default page when any domain was not found (`www.iab.org/documents/docs/2003-09-20-dns-wildcards.html`).

Wildcards can be very confusing in the DNS specifications because of the normal sense in which wildcards are used in search expressions to find items with imprecisely known information. For example, the command `ls |grep $*.html` will list all files in the given directory ending with `.html`; in this case, the * means any character any number of times. In the case of zone files, the wildcard *creates* records of the RR type they are used with such that any query for a particular RR type and for which an explicit RR does *not* exist will be answered with the wildcard RR data as if it *did* exist—that is, no query for the given RR of that type will fail. This may not be the intended result. While wildcards may be used with any RR type, they are most commonly used with MX records as shown in the following zone file fragment:

```
; zone file for example.com
$TTL 2d ; zone default = 2 days
$ORIGIN example.com.
@               IN   MX  10  mail.example.com.
*               IN   MX  10  mail.example.com.
```

In the preceding fragment, an MX query for `bill.example.com`, `joe.example.com`, and `everythingelse.example.com` will return the host `mail.example.com`. The following example shows how an existing RR will block the operation of the wildcard:

```
; zone file for example.com
$TTL 2d ; zone default = 2 days
$ORIGIN example.com.
@               IN   MX  10  mail.example.com.
*               IN   MX  10  mail.example.com.
subdomain   IN  MX  10  mail.example.net.
```

As before, MX queries will return `mail.example.com` except queries for `subdomain.example.com`, which will return `mail.example.net`, and any undefined names below `subdomain` such as `bill.subdomain.example.com` will return `NXDOMAIN` (no name).

A wildcard cannot do anything that cannot be done by one or more (perhaps many more) RRs. There is no *essential* reason to use wildcards other than to reduce the amount of data that may otherwise have to be defined. Whether this reduction in administrative effort is worth the potentially confusing effect of using wildcards in RRs is a matter for local decision.

Summary

This chapter covered a number of common name server configurations and also illustrated some more subtle uses of the DNS system.

The next chapter describes the use of various DNS diagnostic tools and techniques to cover the situations where head-scratching fails to yield the required results.