CHAPTER

The Domain Name System

19

What You Will Learn

In this chapter, you will learn how DNS gives the Internet a more user-friendly way to access resources. We'll see how names are associated with IP addresses and how applications find this information.

You will learn how DNS servers provide information about local networks, and how this information is distributed and shared on the Internet. We'll also use show tools to help examine DNS.

The Domain Name System (DNS) is the distributed database used by the TCP/IP protocol suite to translate hostnames to IP addresses (both IPv4 and IPv6) and provide related information, such as email routing information. DNS has been around as part of the Internet for so long that it is easy to forget that in the early days users needed a file named `/etc/hosts` (no extension) unless they wanted to type in the 32-bit IP address that went along with the hostname.

Today, the database is distributed because no single site on the Internet knows everyone's hostname and IP address. Of course, placing every host's IP address in a single text file would be impractical now, but people can still type `www.juniper.net` anywhere on the Internet and access the main Web page for the site. The correct functioning of DNS is so ingrained in expectations that many users do not even realize that when DNS fails typing, `http://207.17.137.68` yields the same result as the `www` entry. For many, when DNS disappears the Internet might as well have vanished as well (except for some local and cached IP addresses, this is probably true enough).

Microsoft support services report that well over 70% of all calls, no matter what the reported symptom, end up being DNS calls. How can something as apparently simple as DNS cause such problems? Two big reasons are that the details of DNS functioning have changed a lot recently, and that many users and administrators know very little about the inner workings of DNS.

Because of the abundance of new terminology, special operations, and new types of servers, this chapter requires us to discuss some of the basics of DNS before looking

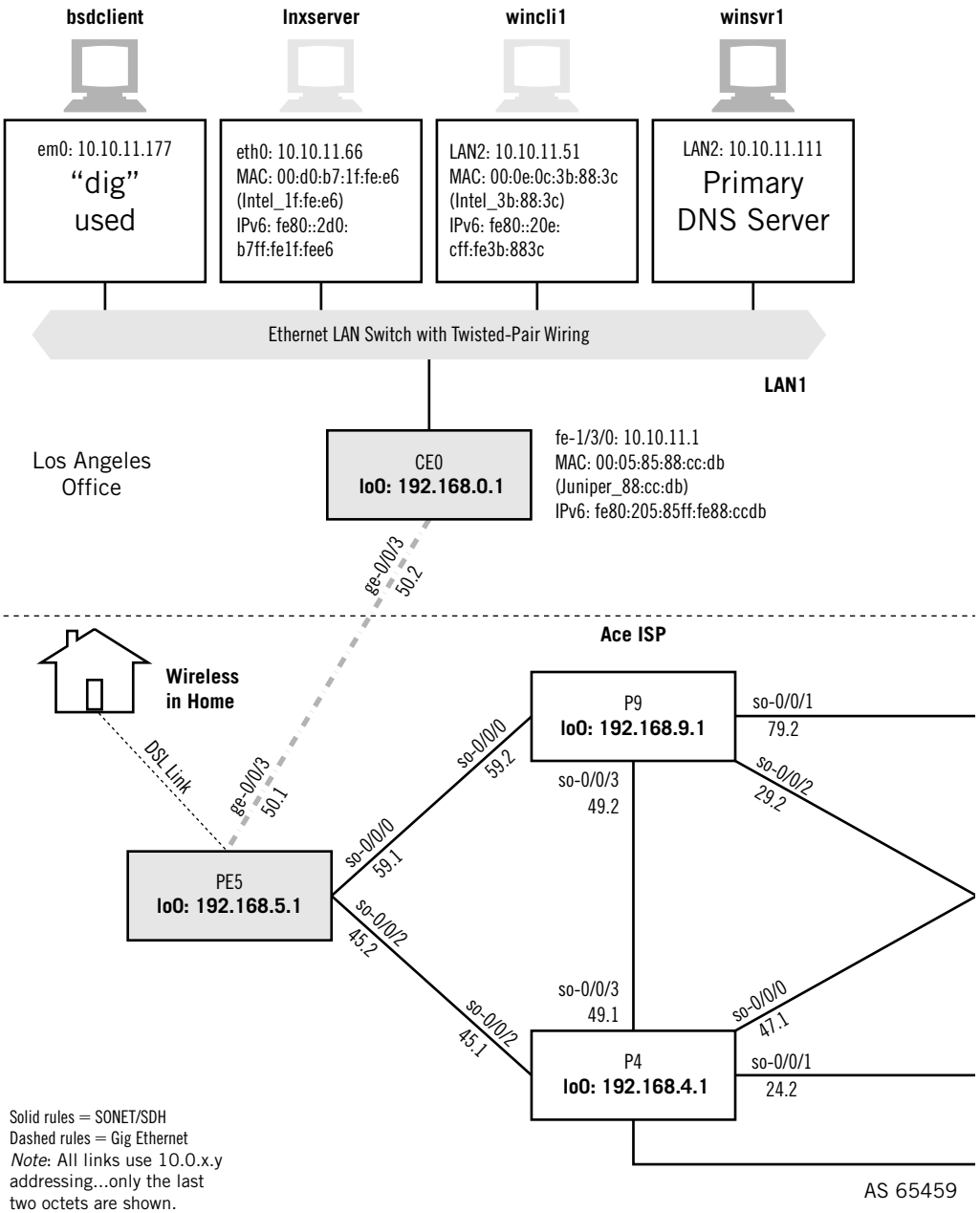
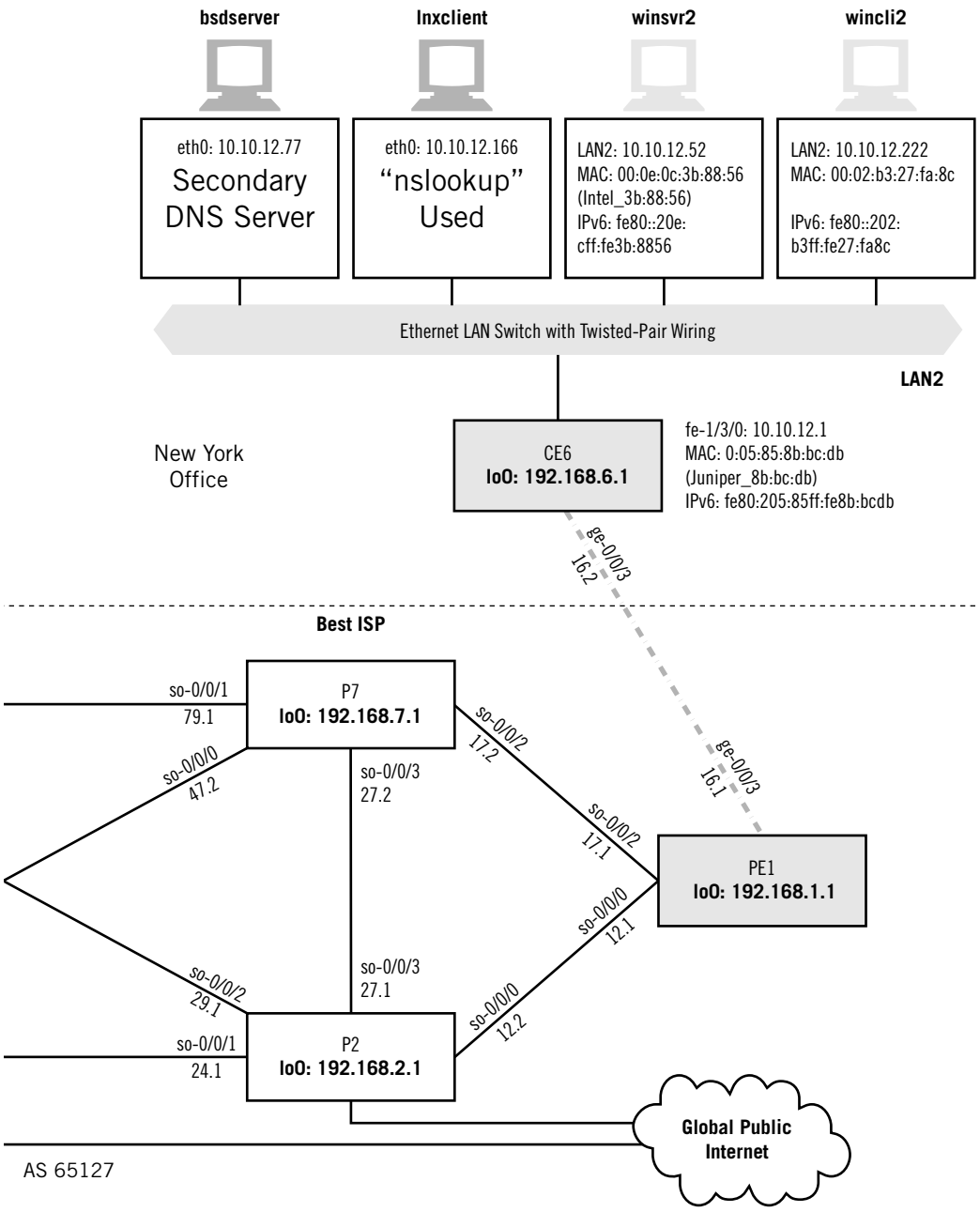


FIGURE 19.1

DNS on the Illustrated Network, showing the hosts used as primary and secondary DNS servers and utilities.


bsdserver
lnxclient
winsvr2
wincli2
eth0: 10.10.12.77
Secondary
DNS Server
eth0: 10.10.12.166
"nslookup"
Used
LAN2: 10.10.12.52
MAC: 00:0e:0c:3b:88:56
(Intel_3b:88:56)
IPv6: fe80::20e:
cff:fe3b:8856
LAN2: 10.10.12.222
MAC: 00:02:b3:27:fa:8c
IPv6: fe80::202:
b3ff:fe27:fa8c
Ethernet LAN Switch with Twisted-Pair Wiring
LAN2
New York
Office
CE6
lo0: 192.168.6.1
fe-1/3/0: 10.10.12.1
MAC: 0:05:85:8b:bc:db
(Juniper_8b:bc:db)
IPv6: fe80:205:85ff:fe8b:bcdb
ge-0/0/3
16.2
Best ISP
so-0/0/1
79.1
P7
lo0: 192.168.7.1
so-0/0/2
17.2
so-0/0/0
47.2
so-0/0/3
27.2
ge-0/0/3
16.1
so-0/0/2
17.1
PE1
lo0: 192.168.1.1
so-0/0/0
12.1
so-0/0/3
27.1
so-0/0/2
29.1
so-0/0/0
12.2
so-0/0/1
24.1
P2
lo0: 192.168.2.1
Global Public
Internet
AS 65127

at how DNS is employed on a network. In this chapter, we'll use the equipment in the roles shown in Figure 19.1. Discussion will be kept to a minimum and exploration is maximized in this chapter.

DNS BASICS

Recall that two things are globally administered in TCP/IP: the network portion of the IPv4 or IPv6 address and the domain name that goes along with it. The host portion of the IP address and the further qualification of the domain name are administered locally. It is up to the local administrator to prevent duplicates at this level, and in large organizations this is not as easy as it sounds. (In some cases there are valid reasons for duplicates to exist in an organization, such as due to "split horizon" issues.) Very large organizations often depend on several layers of administration (perhaps division, department, and so on) to dole out blocks of addresses and domain names correctly. Along with this responsibility goes the duty to ensure that all of the detailed host addressing and the corresponding fully qualified domain names (FQDNs) is correct so that all of the clients can find the servers they are supposed to find.

Usually each site—whether it be a company, university, or other type of organization—maintains its own database of information and runs a server process (typically on a dedicated system) other systems can query. You can also get a third party (not the ISP) to manage a zone for you, and that is a service most registrars will do for a nominal fee (if not free) with the registration of a domain name.

At one time, connection to the Internet required an organization to provide at least two DNS servers for the site. The goal was resilience, but because missing authoritative name serves can cause all sorts of performance issues two non-topologically diverse name serves do not really solve anything. Now, very small organizations (or individual users) often rely on their ISP to provide the DNS service and point all of their hosts at these two "public" DNS servers for hostname resolution. This arrangement poses its own set of problems, such as a recurring ISP charge to "maintain" the database records (surely the lowest maintenance task on the Internet) and the need to update the ISP's database when changes to FQDN or IP addressing take place on the local network. Dynamic IP addresses also cause problems for DNS, as detailed later in this chapter.

The DNS Hierarchy

DNS servers are arranged in a hierarchical fashion. That is, the hundreds of thousands of systems that are *authoritative* for the FQDNs in their *zone* are found at the bottom of the DNS "pyramid." For ease of maintenance, when two or more DNS servers are involved only one of them is flagged as the primary server for the zone, and the rest become secondary DNS servers. Both are authoritative for the zone. ISPs typically run their own DNS servers, often for their customers, with the actual number of systems for each ISP depending on the size of the ISP. At the top of the pyramid is the "backbone." There are root servers for the root zone and others for `.com`, `.edu`, and so on.

DNS servers above the local authoritative level refer other name servers to the systems beneath them, and when appropriate each name server will cache information. Information provided to hosts from any but the authoritative DNS system for the domain is considered *non-authoritative*, a designation not reflecting its reliability, but rather its derived nature.

Authoritative and non-authoritative servers can be further classified into categories. Authoritative servers can be:

- *Primary*—The primary name server for a zone. Find its information locally in a disk file.
- *Secondary*—One or more secondary name servers for the zone. They get their information from the primary.
- *Stub*—A special secondary that contains only name server data and not host data.
- *Distribution*—An internal (or "stealth server") name server known only by IP address.

Keep in mind that the primary and secondary distinction is relevant only to the operator of the systems and not to the querier, who treats them all the same. Non-authoritative servers (technically, only the *response* is non-authoritative) can be:

- *Caching*—Contain no local zone information. Just caches what it learns from other queries and responses it handles.
- *Forwarder*—Performs the queries for many clients. Contains a huge cache.

Root Name Servers

The root servers that stand at the tip of the DNS pyramid deserve more explanation in terms of operation and organization. Today, the root servers are the entry points to the DNS service and rely more on caching than the passive databases that once characterized the root server system. With the explosion of the Internet, it made little sense to maintain records with the same "priority" for sites that are constantly bombarded with traffic and those that are seldom visited. The current database in a root name server is small.

The current root servers only know which name server a local DNS needs to ask next to resolve a query. So, any query for a `.com` sent to a root name server produces a list of name servers that might know the answer. The continuous caching of these answers means that there is less need to query the root servers after the first query.

Root Server Operation

The root server operators are not involved in the policymaking regarding Internet names and addresses, nor in modification of the data. They just take what is originated by one of their number (Verisign Global Registry Services) and propagate it to the others. The operators are encouraged to explore diversity in organizational structure,

locations, hardware, and software, while maintaining expected levels of physical system security and over-provisioning of capacity. They maintain their own infrastructure for emergencies, including telephone hotlines, encrypted email, and secure credentials. The root servers use distributed *anycast* where practical, making many separate systems all over the world appear and act as one system with one IP address. The use of anycast helps minimize the effects of denial-of-service attacks.

We haven't talked about anycast before. In *anycast*, as in multicast, there is a one-to-many association between addresses and *destinations* (multicast has *groups*) on the network. Each destination address identifies a set of receiver endpoints, but (in contrast to multicast) only *one* of them (determined to be the "nearest" or the "best") is chosen at any particular time to receive information from a particular sender. For example, in contrast to a broadcast (which goes to everyone) or a multicast (which goes to all interested listeners) sent onto a LAN, a message to an anycast address goes to only one of a set of hosts and is then considered delivered. Anycast ("send this to any one of these") is more suited to connectionless protocols (such as UDP) than stateful protocols (such as TCP) that have to maintain state information.

Root server operators often struggle to overcome a lot of misconceptions, even on the part of people who should know better. Contrary to what some believe, all Internet traffic does *not* flow through the root servers (nor do they determine routes), not every DNS query goes to a root server, the "A" system is not special, and there *are* many more than just 13 *machines*.

Table 19.1 DNS Root Servers Listed by Operator, Locations, and IP Address

Server	Operator	Locations
A	Verisign	Dulles, VA
B	Information Sciences Institute	Marina Del Rey, CA
C	Cogent Communications	Herndon, VA; Los Angeles; New York; Chicago
D	University of Maryland	College Park, MD
E	NASA Ames Research Center	Mountain View, CA
F	Internet Systems Consortium, Inc.	43 sites all over the world
G	U.S. DoD NIC	Vienna, VA
H	U.S. Army Research Lab	Aberdeen, MD
I	Autonomica/NORDUnet	31 sites all over the world
J	Versign	41 sites all over the world
K	Réseaux IP Européens–Network Coordination Center	17 sites all over the world
L	ICANN	Los Angeles; Miami
M	WIDE Project	6 sites around the world

Root Server Details

Table 19.1 shows the 13 root name servers (A through M), who operates them, their locations, and their IP addresses (IPv4 and IPv6, where applicable). For the latest information, which changes from time to time (for example, the IPv4 address of `B.root-servers.net` changed in 2004), see *www.root-servers.org*.

Note that many of the root servers, although all grouped under a single name, are actually many systems spread throughout the world. This is where anycast is useful.

In the past, the willingness of DNS servers to accept updates from any source when offered was a major security weakness. Modern DNS servers accept only authorized and digitally signed updates, and higher level DNS servers never accept dynamic updates from anyone. One interesting initiative is the continuing development of DNS Security (DNSSec). DNS is still a tempting target on the Internet, and although DNSSec raises the bar the target remains attractive.

DNS IN THEORY: NAME SERVER, DATABASE, AND RESOLVER

DNS consists of three essential components: the name server, the database of DNS resource records, and the resolver. An application interacts with name servers through a resolver. This is an application program that resides on user workstations and sends requests for DNS information when necessary. Resolvers must be able to find at least one name server, usually the local name server, and local DNS servers provide authoritative answers for local systems. The resolver must also be able to use the information returned by the local name server, if the resource records needed are not local or cached, to pursue the query using referral information leading to other DNS name servers on the Internet.

The resource records of the Domain Name Space are grouped and formatted with a strict tree-structured name space. Information is associated with each type of resource record. The sets of local information (the zones) in this structure are distributed among all DNS servers. The name servers essentially answer resolver queries using the information in its zones or from other zones. A resolver query gives the name of interest and stipulates the type of information needed.

The name servers themselves maintain the structure of the Domain Name Space and the sets of information about the hosts in the zones. Any name server can cache anything it sees about any part of any Internet domain, but generally a particular name server knows only about a tiny fraction of the Internet zones. But there are pointers to other name servers that can be used to answer a resolver query. Name servers can distribute zone information to other name servers to provide redundancy. Finally, DNS name servers periodically refresh their zone information, from local files (the primary) or from other name servers (the secondaries) through a zone transfer.

Other important DNS concepts are relative name and absolute name (FQDN). A resolver request for the IP address for the relative name Web server would produce many addresses on many networks around the world. The relative name is part of the complete absolute name, perhaps *webserver.example.com*. Most resolvers step

through an ordered list of preconfigured suffixes, append them one at a time to the relative name, and attempt to find the IP address without the absolute name. Absolute names always end in a dot (.).

Like all good protocols using query/response pairs, DNS uses UDP (port 53). However, DNS also uses TCP (and port 53 there, too) for zone transfers between name servers. These transfers can be considerable in large organizations, and although LANs usually feature very low-error rates the risk of corrupt DNS information more than justifies the use of TCP for the zone transfers. TCP is also used if a response is larger than 512 bytes. And flow control is a really good reason to use TCP for zone transfers, because they can occur over essentially arbitrary distances.

Adding a New Host

Whenever a new host is added to a zone, the DNS administrator must add the resource records (minimally the name and IP address of the host) to a file on the primary name server. The primary name server is then told to read the configuration files, and when the secondaries query the primary (typically every 3 hours), the secondaries find newer information on the primary and perform a zone transfer. The DNS Notify feature enhances the basic zone status check and zone transfer mechanisms. This lets the primary server notify the secondaries when the database has changed. A related feature allows part of a zone to be transferred and not the entire zone information.

How can all of the local name servers find each other? They can't. But every name server must be able to find and contact the root name servers on the Internet. Their positions at the top of the DNS pyramid allow the root name servers to answer queries directly from the zone they have loaded, if with non-authoritative information. Of course, there's always a chance a user on one side of the world will attempt to contact a server or Web site that has just been linked to the Internet and has the zone information such as the IP address available only in the local name server on the network with the Web site.

Recursive and Iterative Queries

If DNS database information is spread throughout the Internet, and the local name servers cannot find each other and the root name servers don't have gigantic databases, how can all hosts in the world find out anything at all? It is because of the way the local DNS name server handles a query from a resolver.

DNS queries can be sent out asking for another name server to handle the query recursively or iteratively (some texts say "non-recursively"). Most local DNS servers function recursively by default. In fact, recursive operation maximizes the amount of information available for caching on name servers, although iterative operation will maximize the amount of information available to a particular name server. Many local name servers use recursive queries (they can be asked to handle a query iteratively), and higher level name servers use iterative queries (root servers always answer queries iteratively).

Recursive DNS queries are handled by the receiving name server waiting until it receives an answer to its own queries. Iterative queries are handled with an immediate "I don't know the answer, but here's where you can look next" response. In the recursive case, the name server "in the middle" can find and cache the information, whereas in the iterative case, it cannot. This might sound confusing, but we'll look at a detailed example of how DNS usually works in the following sections.

Delegation and Referral

Large organizations, or large ISPs operating the DNS servers for their customers, often delegate part of the domain name space to a separate system. For example, a huge *bigcompany.com* might have headquarters records on the main DNS but delegate DNS chores for maintaining and housing *east.bigcompany.com* (on the east coast) and *west.bigcompany.com* (on the west) to its two main divisions. So, there are three DNS servers in all, perhaps called *hqns.bigcompany.com, ns1.east.bigcompany.com* on the east coast and *ns2.west.bigcompany.com* on the west coast. There could be many LANs for which one of these name servers is authoritative, such as the LANs for accounting, marketing, sales, and so on.

Figure 19.2 shows the flow of DNS-related actions (solid arrows) and the responses they invoke (dashed arrows) among the DNS name servers mentioned in the *bigcompany.com* example the first time someone looks for the Web site. The initial user resolver query to the LAN's local name server and the eventual response are also shown. The following is the sequence in detail.

The local user on the `wincli1` Web browser (me) requests a Web page from *www.sales.west.bigcompany.com* (the example is valid, but the name has been changed). The browser invokes the local name resolver software in the PC and passes this name to it.

The local resolver checks its cache to see if there is already an IP address stored for this name. (If there is, the quest is over, but we've assumed that this is the first time the user has asked for the Web site so it's not cached.) The resolver also checks to see if there is a local host table file. (Again, let's assume there is no static mapping for the name.)

The resolver generates a recursive query (typically) and sends it to the local name server, which we've set up as `ns1.booklab.englab.jnpr.net` on `winsrv1` using the name server's IP address, which it knows because the server is local (it's `10.10.11.111`). The local DNS system receives the request and checks its cache. If present, the DNS returns a non-authoritative response to the resolver. It would also check to see if there are zone resource records for the request name, but because they are completely different domains there are no zone records.

The local DNS generates an iterative request containing the name sought and sends it to a root name server. The root name server doesn't resolve the name, but returns the name and IP address of the name server for the `.com` domain. The local DNS (which is performing the bulk of the work, we should note) now sends an iterative request to the name server for the `.com` domain.

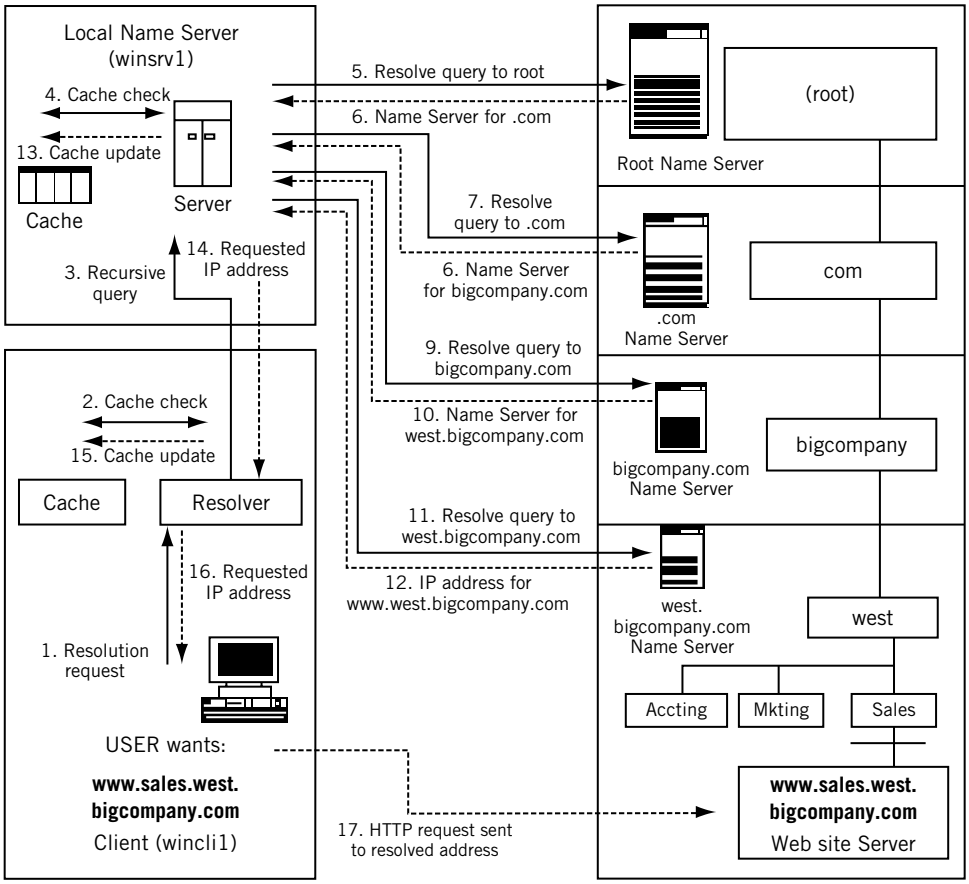


FIGURE 19.2

Example DNS query and response message flow. Messages sent to the servers are shown as solid arrows and replies as dashed arrows.

The `.com` name server returns the name and IP address for the name server for the `bigcompany.com` domain. The local DNS then generates an iterative request to the name server for the `bigcompany.com` domain. The `bigcompany.com` name server looks to see if it has that information. It notices that the requested name is in a separate zone, the `west.bigcompany.com` subdomain.

The local DNS next generates an iterative request to the name server for the `west.bigcompany.com` domain. This name server is authoritative for the *www.sales.west.bigcompany.com* information. It returns the address information for the host to the local DNS. The local DNS system (`winsrv1`) caches the information.

The local DNS returns the resolution to the client's resolver software (`wincli1`). The local resolver also caches the information. The local resolver supplies the address information to the browser. The browser can now send an HTTP request to the correct IP address.

It's actually a tribute to the entire DNS server collection that all of this usually happens very quickly. Note how using recursion on the PC maximized the amount of DNS information available for caching and how iteration elsewhere minimized the amount of information needing to be stored permanently.

Glue Records

There was one key step in the chain of delegation and referral in Figure 19.2 that did not use DNS to find an IP address. Notice that the `bigcompany.com` name server did not use DNS to find the IP address of the `west.bigcompany.com` name server. Delegation must use an address (A) resource record to indicate the IP addresses of name servers responsible for zones below the current level. These are called glue records in DNS and are the answer to an interesting question involving dynamic IP address allocation.

When DHCP first became available, many organizations configured a pool of IP addresses to be assigned only to active users on the Internet. Many organizations included their DNS servers in this pool, and quickly found out that DNS stopped working. Why? Simply, the glue records used by intermediate name servers to find the local authoritative servers didn't work anymore. In other words, the headquarters can't use DNS to find the zone resource records for delegated zones! Glue records serve that purpose.

This is one main reason users whose ISPs use DHCP with dynamic IP addresses for host configuration cannot establish their own DNS name server at home. These users would form delegated zones from the main ISP. And without a local DNS server users who want to place their own server on-site need to work with the ISP to make this happen. Some people see this as part of an ISP plot to prevent users from running their own servers, creating hosting revenue for ISPs and others. But it's really just the glue records.

You need a DNS service provider willing to upgrade the glue records when your address changes. In practice, dynamic DNS service providers can do this, but it also means that the TTL on the records must be low enough so that they flow over in short order. Ideally, they would also provide a secondary DNS.

DNS IN PRACTICE: RESOURCE RECORDS AND MESSAGE FORMATS

When implemented as a series of resolvers and name servers, DNS databases consist of resource records (RRs) entered into a zone file and loaded onto the authoritative name server. Any other DNS name server can cache this information as a non-authoritative source, and a special reverse zone file is used to enable resolvers to look up a host name by IP address. RRs all end in in-addr.arpa. A DNS RR contains the following fields.

Name—The FQDN or portion that is represented by the entry. For example, `bigcompany.com`.

TTL (Time to Live)—How long in seconds the record can be cached. Many ISPs use 2 or even 3 days for this field (172,800 or 259,000). If no value is entered, the default can be short (as little as 1 hour).

Class—Today, the only class that counts is `IN` for Internet address. This is usually entered only once, in the first record, and is inherited by all subsequent records for that name.

Record-Type—There are many record types, usually indicated by a short abbreviation, such as `A` for address and `NS` for name server. The types fall into four categories:

Table 19.2 Common DNS Resource Record Types and Their Uses and Meanings

Use	Record Type	Meaning
Zone	SOA	Start of Authority records identify the zone and set parameters.
	NS	Gives an authoritative name server for the zone, and delegates subdomains. Not the IP address of the name server, but a text field.
Basic	A	Maps the name to the IPv4 address. Each device address requires a separate A record.
	AAAA	Used to allow an IPv4 name server to return an IPv6 address. Intended as a transitional type.
	A6	Now obsolete, these were used to map a name to an IPv6 address.
	PTR	Used to map an IP address to a host name in reverse zone lookups.
	DNAME	Formerly used for redirection for reverse lookups in IPv6 DNS servers due to longer nature of IPv6 addresses. Now obsolete.
	MX	Mail Exchanger records point from a name to A records that are the mail exchanger for the name.
Security	KEY	The public key for the DNS name.
	NXT	Used for negative answers with DNSSec.
	SIG	The signature for an authenticated zone.
Optional	CNAME	Maps an alias name to a canonical ("real") name. For example, www.example.com and ftp.example.com might both be running on the host server.example.com.
	LOC	Geographical location.
	NAPTR	Name Authority Pointer is used to allow regular expression rewrites of the domain name.
	RP	Contact information for responsible person.
	SRV	Gives locations of well-known services.
	TXT	To add comments and information to the record.

zone, basic, security, and optional. A list of the more common record types appears in Table 19.2.

Record-Data—Depending on the type, this information varies. For a name server, this is the domain name of the name server. For a host, this is the IP address.

Comments—These are optional and begin with a semicolon (;) and are never returned with data.

This is not an exhaustive list. Some defined record types are seldom used (HINFO is supposed to mention host model and operating system) or are perceived as security risks (WKS records list the "well-known services" available at the host).

Some readers might have noticed the elaborate form of the IPv6 addresses used on the Illustrated Network. This is because IPv6 once used something called the binary label syntax. IPv6 addresses use the first bits (really, whole words) of the 128-bit IPv6 address to indicate the ISP. The A6 records included a referral field to allow a name server to refer to the ISP's name server for the "network" portion of the IPv6 address. The A6 record also gave the number and value of the bits present in the A6 record itself. This prevented the laborious entry of many redundant bits into the resource records. It also made shifting service providers easier. So, a query for an A6 record might only get the last 64 bits of an IPv6 address. A further referral query to the name server in the A6 record is necessary for the first 64 bits. The DNAME records do the same for the Pv6 host name. This now obsolete system was used for the IPv6 addresses.

The same DNS message format is used for queries and responses. The DNS query message goes out with a 12-octet header and a variable number of questions. The DNS response message essentially pastes on a variable number of three types of response fields: answer RRs, RRs identifying authoritative servers, and RRs with additional information. Figure 19.3 shows the general format of the DNS message.

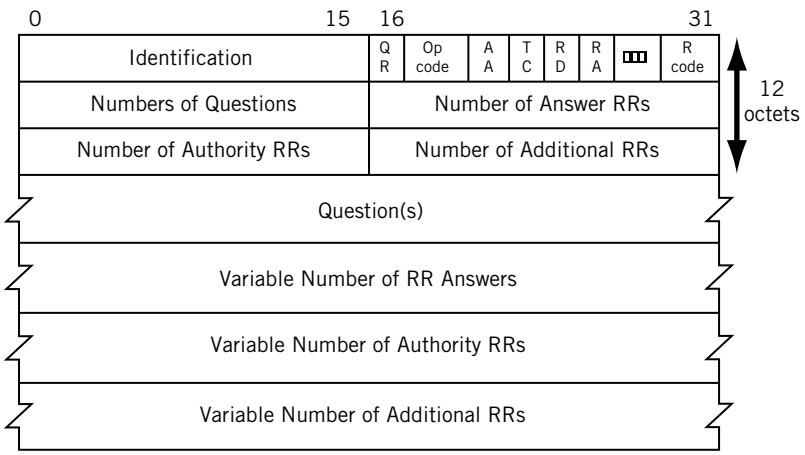


FIGURE 19.3

DNS message format. Note that the last four fields are variable in size.

DNS Message Header

The 16-bit identification field, set by the client and returned by the server, allows for coordination of outstanding requests and responses. The 16-bit Flags field is quite complex:

QR—A 1-bit field where 0 = query and 1 = response.

Opcode—A 4-bit field where 0 = standard query. Other values are for an inverse query (1) and a server status request (2).

AA—A 1-bit flag that indicates that the name server is authoritative for the zone (1 = true).

TC—A 1-bit flag meaning that the reply has been truncated. UDP limits DNS responses to 512 octets, except when Extension Mechanism for DNS (`EDNS0`, defined in RFC 2671) is used. EDNS0 identifies the requester's UDP packet size.

RD—A 1-bit flag for "recursion desired." If this bit is set in a query, the receiving name server is supposed to keep trying to find the answer. If this bit is not set, the name server returns a list of other name servers to contact unless it can provide an authoritative answer.

RA—A 1-bit flag for "recursion available." Some name servers will refuse to act recursively, and this bit is cleared in response to let other systems know about server refusal.

Pad—A 3-bit field that must be set to 000.

Rcode—A 4-bit field for the return code. The most common values are for no error (0) and a name error (3).

The next four 16-bit fields help receivers parse the four fields in the rest of the message. In a query, the number of questions is usually 1 and the other three fields are 0. A reply typically sets the Number of Answers field to 1 (or more), and the other two are 0. Utilities such as tcpdump and Ethereal normally parse all of the fields and flags. There are other ways to watch DNS in action, however.

DNSSec

As indispensable as DNS is for Internet operation, DNS was not (unfortunately) designed to be secure. Threats to DNS fall into several distinct classes, many of which are just well-known security threats redirected at DNS. However, a few are specific to the particular way the DNS protocol functions. RFC 3833 documents some of the known threats to DNS and tries to assess the extent to which DNSSec will succeed in defending against these threats. Although this section uses some concepts we haven't covered yet, DNSSec is important enough to introduce in this chapter on DNS itself.

In particular, DNSSec was designed to protect Internet DNS resolvers (the clients) from forged DNS data, which can point people looking for a particular Web site (such as their bank) to the wrong IP address. This forged information can be put in place by a process called DNS cache poisoning. In DNSSec, all answers to queries are digitally signed (we'll talk more about digital signatures and certificates in Chapters 22 and 23). The digital signature can be checked by the resolver to see if the information is identical to the information on the authoritative DNS server for the site. DNSSec, although designed primarily to protect IP addresses, can be used to protect other information (such as the cryptographic certificates stored in DNS). RFC 4367 describes how to use DNS to distribute certificates, including those used for email, so it is possible to use DNSSec as a global infrastructure for secure email.

However, DNSSec does not say anything about the confidentiality of data. That is, all DNSSec responses are authenticated but not encrypted (we'll talk more about the differences in Chapter 29). It also really doesn't protect against denial-of-service attacks directly, although DNSSec does provide some benefit through the authentication features of the digital signature. Other methods must be used to protect bulk data, such as a large zone transfer. Of course (per RFC 4367) DNSSec cannot prevent users from making false assumptions about domain names, such as the idea that the organization's name plus *.com* is always the company (or bank) Web site they are looking for. But at least DNSSec can authenticate that the data provided by DNS is actually from the domain owner.

The current DNSSec specifications describe DNSSec-bis. The most important are RFC 4033, RFC 4034, and RFC 4035.

DNS Tools: nslookup, dig, and host

The Berkeley Internet Name Domain (BIND), developed for the Unix environment, is both resolver and name server. When BIND is running as name server, the process is named. Entire books have been written about DNS and BIND, so this chapter can only look at a few of the things that can be explored with a few simple DNS tools and utilities.

BIND configuration statements for a zone are in named.conf, usually found in `/etc`—where the name servers to be contacted (in `resolv.conf`) are also located. A "hints" file (variously named *named.ca, named.root,* or *root.cache*) has information about the root servers and essentially "primes" the DNS cache at start-up.

The nslookup utility program allows a user to interact with a DNS name server directly. Options allow the user to display detailed query and response information as needed. Originally a testing tool, nslookup functions in both interactive and non-interactive mode. Today, the use of nslookup is deprecated, and it is not included in many operating system distributions. Its functionality has been taken over by dig and host.

The Domain Internet Groper (dig) DNS query tool is more general than nslookup, and is often used with other tools. It has a consistent output format that is easily parsed with other programs, and is available for Windows 2000/XP (but not 98/ME).

Over time, dig developed a distinct "feature sprawl" that offended some who favored clean and mean Internet tools. The host utility by Eric Wassenaar is intended to be an evolutionary step for both nslookup and dig. The examples in this chapter will use dig as well as nslookup, if only because of the familiarity of the nslookup format.

DNS IN ACTION

Putting a functioning DNS system on the Illustrated Network will allow us to do things such as ping *winsrv1.booklab.englab.jnpr.net* instead of having to know the IP address and use ping `10.10.11.111`. We'll go against common wisdom and make a Windows XP system (`winsrv1`) our primary DNS server, and we will use the FreeBSD server (`bsdserver`) as the secondary DNS for LAN1 and LAN2. Windows XP Pro does not support DNS natively, so we'll use a GUI-based DNS server package called SimpleDNS instead of BIND.

Once DNS is up and running, we have to ensure that all hosts know where to find it. On `lnxclient`, and most Unix hosts, we just add them to the `/etc/resolv.conf` file.

```
search booklab.englab.jnpr.net englab.jnpr.net jnpr.net
nameserver 10.10.11.111
nameserver 10.10.12.77
```

Now, let's see how DNS works to find local hosts.

```
[root@lnxclient admin]# nslookup
Note:  nslookup is deprecated and may be removed from future releases.
Consider using the 'dig' or 'host' programs instead.  Run nslookup with
the '-sil[ent]' option to prevent this message from appearing.
> winsrv1
Server:         10.10.11.111
Address:        10.10.11.111#53

Name:   winsrv1.booklab.englab.jnpr.net
Address: 10.10.11.111
> winscli1
Server:         10.10.11.111
Address:        10.10.11.111#53

Name:   wincli1.booklab.englab.jnpr.net
Address: 10.10.11.51
> bsdserver
Server:         10.10.11.111
Address:        10.10.11.111#53

Name:   bsdserver.booklab.englab.jnpr.net
Address: 10.10.12.77
>
```

Note the "warning" about continued use of nslookup. But it still works. Of course, if we pause the DNS on `winsrv1`, we can still get a response from `bsdserver` (as long as a zone transfer has taken place).

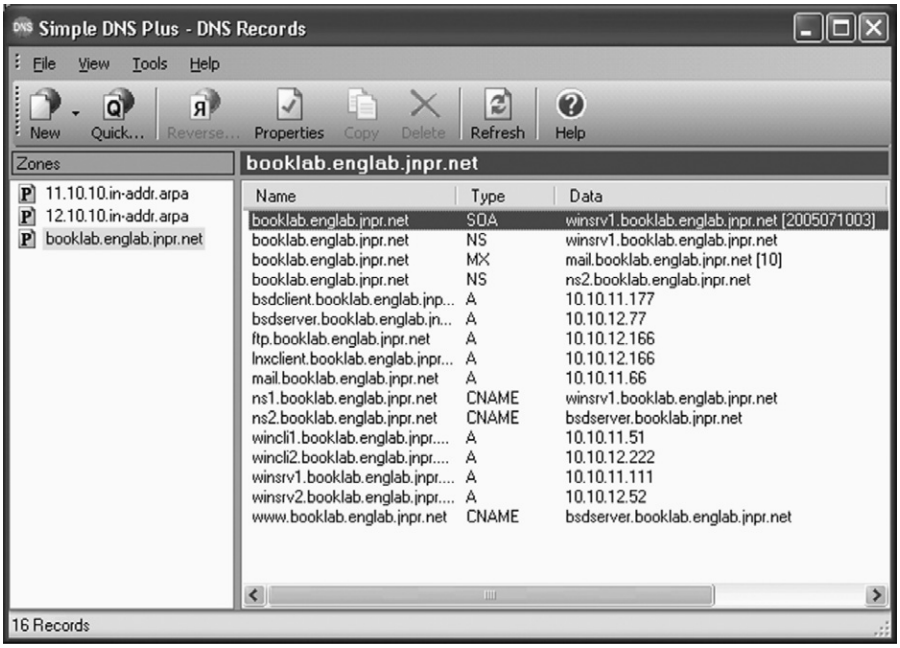


FIGURE 19.4

DNS records on `winsrv1` using a GUI. Note the various record types (the name servers in particular).

```
> lnxserver
Server: 10.10.12.77
Address: 10.10.12.77#53

Non-authoritative answer:
Name: lnxserver.booklab.englab.jnpr.net
Address: 10.10.11.66
```

Simple DNS has a nice GUI, in contrast to the text files used in most Unix DNS versions (as shown in Figure 19.4).

The Ethereal capture in Figure 19.5 shows the utter simplicity of the DNS message exchanges. There's even a nice log of these messages, as shown in Figure 19.6 (it also tracks DHCP leases when dynamic DNS is used).

Now we can finally `ping` on the Illustrated Network the "normal" way.

```
[root@lnxclient admin]# ping wincli1.booklab.englab.jnpr.net
PING wincli1.booklab.englab.jnpr.net (10.10.11.51) 56(84) bytes of data.
64 bytes from wincli1.booklab.englab.jnpr.net (10.10.11.51): icmp_seq=1
 ttl=126 time=0.768 ms
64 bytes from wincli1.booklab.englab.jnpr.net (10.10.11.51): icmp_seq=2
 ttl=126 time=0.283 ms
```

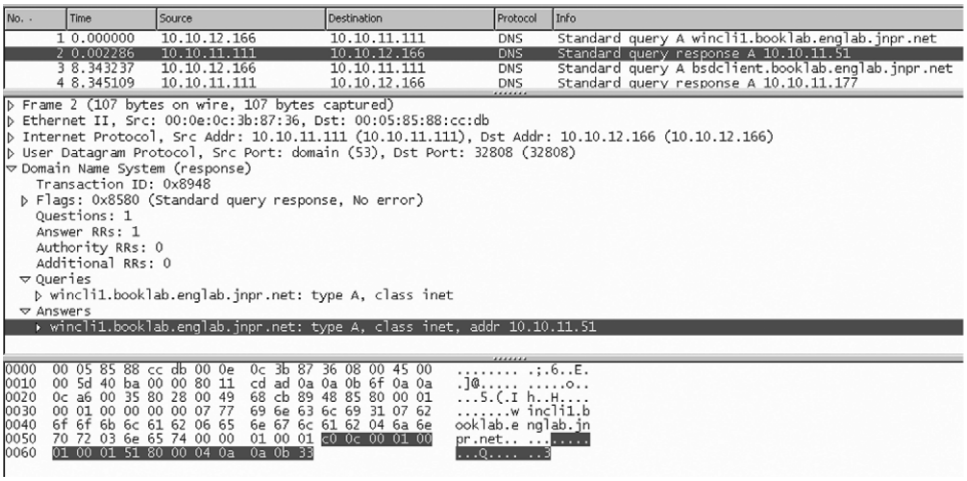


FIGURE 19.5

DNS server reply. Note that the question field shows up as "queries."

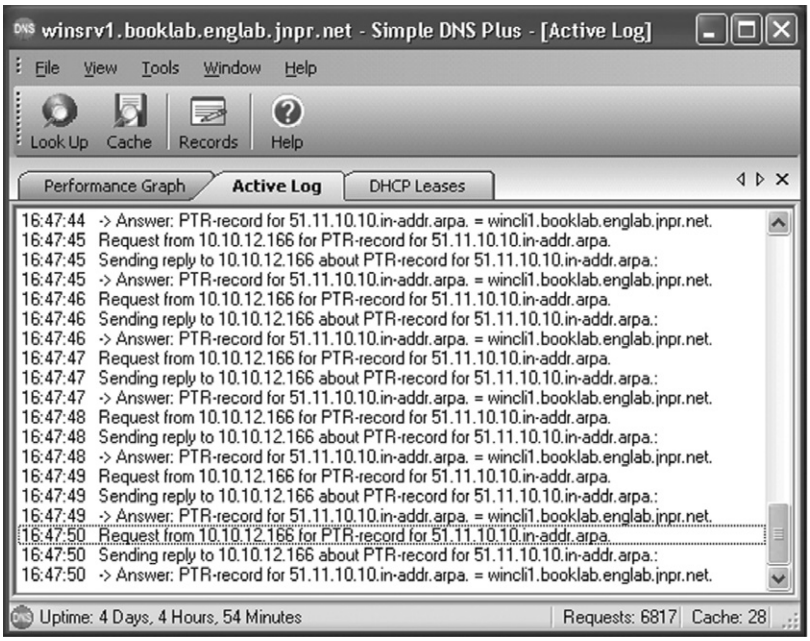


FIGURE 19.6

DNS server log showing the history of queries and responses.

```
64 bytes from wincli1.booklab.englab.jnpr.net (10.10.11.51): icmp_seq=3
 ttl=126 time=0.285 ms
64 bytes from wincli1.booklab.englab.jnpr.net (10.10.11.51): icmp_seq=4
 ttl=126 time=0.259 ms
64 bytes from wincli1.booklab.englab.jnpr.net (10.10.11.51): icmp_seq=5
 ttl=126 time=0.276 ms
64 bytes from wincli1.booklab.englab.jnpr.net (10.10.11.51): icmp_seq=6
 ttl=126 time=0.244 ms
64 bytes from wincli1.booklab.englab.jnpr.net (10.10.11.51): icmp_seq=7
 ttl=126 time=0.259 ms
^C

--- wincli1.booklab.englab.jnpr.net ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 8080ms
rtt min/avg/max/mdev = 0.244/0.325/0.768/0.158 ms
[root@lnxclient admin]#
```

LAN1 is also running a DNS server on `lnxserver`, and to keep the configuration very simple only functions as a non-authoritative server. The configuration is short and sweet:

```
lnxserver$ cat /etc/named.conf
options {
  directory "/var/named";
};
// this is a caching only name server zone configuration
zone "." {
  type hint;
  file "named.ca";
};
zone "0.0.127.in-addr.local";
  type master;
  file "named.local";
};
```

The two zone statements only point to the root servers on the Internet (in the hints file *named.ca)* and make this server the master for its own loopback address. These two zones appear in all name server configurations.

We should also limit the hosts from which recursion can be performed on the caching name server. Otherwise, it might get used as a denial-of-service amplifier. That section would be:

```
allow-recursion { 127.0.0.1;
10.10.11.0/24;
};
```

We'll point to the lnxserver name server on wincli1 on LAN1 and use nslookup to verify that we can still find the Internet name servers. At the interactive DNS prompt (>), we'll set the type of query to send to ns for name servers and we will look for "com."

This is the root of the entire ".com" Domain Name Space (note that we ask for `com.` and not `.com` without the ending dot). Otherwise, the system would append a suffix and try to find *com.booklab.englab.jnpr.net* and return an error (unless we did have a system named "com" on the network).

```
> com.
Server: lnxserver.booklab.juniper.net
Address: 192.168.27.14

Non-authoritative answer:
com    nameserver = f.gtld-servers.net
com    nameserver = g.gtld-servers.net
com    nameserver = h.gtld-servers.net
com    nameserver = i.gtld-servers.net
com    nameserver = j.gtld-servers.net
com    nameserver = k.gtld-servers.net
com    nameserver = l.gtld-servers.net
com    nameserver = m.gtld-servers.net
com    nameserver = a.gtld-servers.net
com    nameserver = b.gtld-servers.net
com    nameserver = c.gtld-servers.net
com    nameserver = d.gtld-servers.net
com    nameserver = e.gtld-servers.net

a.gtld-servers.net   internet address = 192.5.6.30
a.gtld-servers.net   AAAA IPv6 address = 2001:503:a83e::2:30
b.gtld-servers.net   internet address = 192.33.14.30
b.gtld-servers.net   AAAA IPv6 address = 2001:503:231d::2:30
c.gtld-servers.net   internet address = 192.26.92.30
d.gtld-servers.net   internet address = 192.31.80.30
e.gtld-servers.net   internet address = 192.12.94.30
f.gtld-servers.net   internet address = 192.35.51.30
g.gtld-servers.net   internet address = 192.42.93.30
h.gtld-servers.net   internet address = 192.54.112.30
i.gtld-servers.net   internet address = 192.43.172.30
j.gtld-servers.net   internet address = 192.48.79.30
k.gtld-servers.net   internet address = 192.52.178.30
l.gtld-servers.net   internet address = 192.41.162.30
m.gtld-servers.net   internet address = 192.55.83.30
```

There are 13 servers, A through M, on the first part of the list. But instead of being called "root servers" these are "gltd servers." GLTD stands for generic top-level domains (sometimes seen as gTLD), and that's what the traditional Internet host name endings such as `.com`, `.mil`, `.org`, and so on are in DNS. There are also ccTLDs (country code TLDs), such as `.fr` for France and `.ca` for Canada.

Note that the A and B GTLD servers return AAAA record types, showing that the A6 and DNAME records (once so promising) are obsolete. We're not supposed to use nslookup (dig is not built into Windows XP, but can be installed as freeware). Let's see what dig can do, this time on the FreeBSD client.

```
bsdclient# dig

; <<>> DiG 8.3 <<>>
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10624
;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 13
;; QUERY SECTION:
;;    ., type = NS, class = IN

;; ANSWER SECTION:
.                       12h46m16s IN NS  d.root-servers.net.
.                       12h46m16s IN NS  a.root-servers.net.
.                       12h46m16s IN NS  h.root-servers.net.
.                       12h46m16s IN NS  c.root-servers.net.
.                       12h46m16s IN NS  g.root-servers.net.
.                       12h46m16s IN NS  f.root-servers.net.
.                       12h46m16s IN NS  b.root-servers.net.
.                       12h46m16s IN NS  j.root-servers.net.
.                       12h46m16s IN NS  k.root-servers.net.
.                       12h46m16s IN NS  l.root-servers.net.
.                       12h46m16s IN NS  m.root-servers.net.
.                       12h46m16s IN NS  i.root-servers.net.
.                       12h46m16s IN NS  e.root-servers.net.

;; ADDITIONAL SECTION:
d.root-servers.net.     12h46m16s IN A  128.8.10.90
a.root-servers.net.     12h46m16s IN A  198.41.0.4
h.root-servers.net.     12h46m16s IN A  128.63.2.53
c.root-servers.net.     12h46m16s IN A  192.33.4.12
g.root-servers.net.     12h46m16s IN A  192.112.36.4
f.root-servers.net.     12h46m16s IN A  192.5.5.241
b.root-servers.net.     12h46m16s IN A  192.228.79.201
j.root-servers.net.     12h46m16s IN A  192.58.128.30
k.root-servers.net.     12h46m16s IN A  193.0.14.129
l.root-servers.net.     12h46m16s IN A  198.32.64.12
m.root-servers.net.     12h46m16s IN A  202.12.27.33
i.root-servers.net.     12h46m16s IN A  192.36.148.17
e.root-servers.net.     12h46m16s IN A  192.203.230.10

;; Total query time: 1 msec
;; FROM: bsdclient.booklab.englab.jnpr.net to SERVER: 10.10.11.66
;; WHEN: Fri Feb 22 10:10:00 2008
;; MSG SIZE  sent: 17  rcvd: 449

bsdclient#
```

That's a lot more detailed information, and it doesn't use an interactive prompt. By default, dig looks for root NS records and serves up flags, TTL information (in user-friendly units), and so on. Let's look at a more complete (or realistic) example and look

for the IP address of the server for *www.amazon.com* (perhaps so you can prepare to order more copies of this book).

```
bsdclient# dig www.amazon.com

; <<>> DiG 8.3 <<>> www.amazon.com
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10904
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUERY SECTION:
;;     www.amazon.com, type = A, class = IN

;; ANSWER SECTION:
www.amazon.com.         1m7s IN A       207.171.175.35

;; Total query time: 95 msec
;; FROM: bsdclient.booklab.englab.jnpr.net to SERVER: 10.10.11.66
;; WHEN: Fri Feb 22 10:40:17 2008
;; MSG SIZE  sent: 32  rcvd: 48
```

`dig` got us an answer, but not an authoritative one (`AUTHORITY: 0`). To get the authoritative answer to the Amazon Web site, and not something from cache, we'll have to find the Amazon name servers and ask one of them.

```
bsdclient# dig www.amazon.com ns

; <<>> DiG 8.3 <<>> www.amazon.com ns
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44598
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1
;; QUERY SECTION:
;;      www.amazon.com, type = NS, class = IN

;; ANSWER SECTION:
www.amazon.com.         21h7m55s IN NS  ns-40.amazon.com.
www.amazon.com.         21h7m55s IN NS  ns-30.amazon.com.
www.amazon.com.         21h7m55s IN NS  ns-20.amazon.com.
www.amazon.com.         21h7m55s IN NS  ns-10.amazon.com.

;; ADDITIONAL SECTION:
ns-40.amazon.com.       21h7m55s IN A   207.171.169.7
;; Total query time: 1 msec
;; FROM: bsdclient.booklab.englab.jnpr.net to SERVER: 10.10.11.66
;; WHEN: Fri Feb 22 10:38:37 2008
;; MSG SIZE  sent: 32  rcvd: 128
```

Amazon has four name servers (note we found these answers cached, because of the `AUTHORITY: 0`). We'll ask `ns-40` about Amazon's Web site:

```
bsdclient# dig @ns-40.amazon.com www.amazon.com A

; <<>> DiG 8.3 <<>> @ns-40.amazon.com www.amazon.com A
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6717
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
;; QUERY SECTION:
;;     www.amazon.com, type = A, class = IN

;; AUTHORITY SECTION:
www.amazon.com.         1m7s IN A       207.171.166.48

;; Total query time: 3 msec
;; FROM: bsdclient.booklab.englab.jnpr.net to SERVER: 204.74.101.1
;; WHEN: Fri Feb 22 10:32:52 2008
;; MSG SIZE  sent: 32  rcvd: 112
```

Now `AUTHORITY: 1` appears. It's nice to know that Amazon's own name server is authoritative for itself. But let's not get too worried about authoritative answers. Cached information is usually just as good. In fact, look what happens when we repeat the query.

```
bsdclient# dig @ns-40.amazon.com www.amazon.com A

; <<>> DiG 8.3 <<>> @ns-40.amazon.com www.amazon.com A
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52895
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUERY SECTION:
;;      www.amazon.com, type = A, class = IN

;; ANSWER SECTION:
www.amazon.com.         1m7s IN A       207.171.175.35

;; Total query time: 91 msec
;; FROM: bsdclient.booklab.englab.jnpr.net to SERVER: 207.171.169.7
;; WHEN: Fri Feb 22 10:55:29 2008
;; MSG SIZE  sent: 32  rcvd: 48
```

Isn't the `ns-40` server still authoritative? Sure, but our earlier query just popped that information into the local cache. Why fetch up an authoritative reply when there's one just as good in cache? Caching can be a nuisance when trying to "force" authoritative answers, especially across the Internet.

Dig has been criticized for feature bloat. For comparison, the host DNS utility retains the clean and sparse Unix output philosophy.

```
bsdclient# host www.amazon.com
www.amazon.com has address 207.171.166.102
bsdclient#
```

Even at its most verbose, host is not as forthcoming as the other utilities.

```
bsdclient# host -v www.amazon.com ns-40.amazon.com
Using domain server:
Name: ns-40.amazon.com
Addresses: 207.171.169.7

Trying null domain
rcode = 0 (Success), ancount=1
The following answer is not verified as authentic by the server:
www.amazon.com  67 IN   A      207.171.175.29
```

This has been by no means an exhaustive look at how DNS acts. For more information, the excellent *DNS and BIND* by Cricket Liu (O'Reilly Media) should be considered definitive.

QUESTIONS FOR READERS

Figure 19.7 shows some of the concepts discussed in this chapter and can be used to help you answer the following questions.

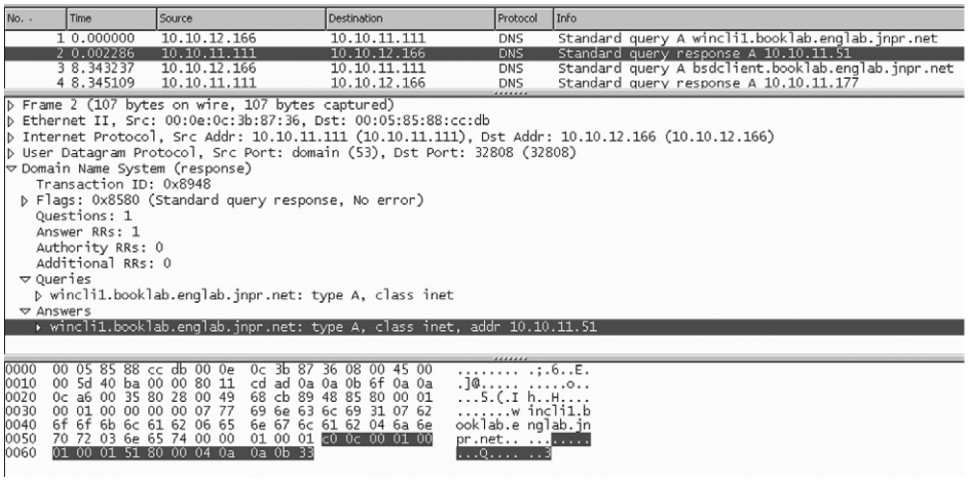


FIGURE 19.7

A DNS server reply message parsed by Ethereal.

1. How many questions (queries) are usually present in a DNS request?
2. Is the message in the figure a query or a response?
3. What are the host names of the client and the DNS server on the Illustrated Network that correspond to the IP addresses in the figure?
4. The flag field value is `0x8580`. Is the DNS server authoritative for the zone?
5. Based on the flag field value, is recursion desired and available?