1
Domain Name System

All applications that provide communication between computers on the Internet use IP addresses to identify communicating hosts. However, IP addresses are difficult for human users to remember. That is why we use the name of a network interface instead of an IP address. For each IP address, there is a name of a network interface (computer)—or to be exact, a domain name. This domain name can be used in all commands where it is possible to use an IP address. (One exception, where only an IP address can be used, is the specification of an actual name server.) A single IP address can have several domain names affiliated with it.

The relationship between the name of a computer and an IP address is defined in the **Domain Name System** (**DNS**) database. The DNS database is distributed worldwide. The DNS database contains individual records that are called **Resource Records** (**RR**). Individual parts of the DNS database called **zones** are placed on particular name servers. DNS is a worldwide distributed database.

If you want to use an Internet browser to browse to `www.google.com` with the IP address 64.233.167.147 (Figure 1.1), you enter the website name `www.google.com` in the browser address field.

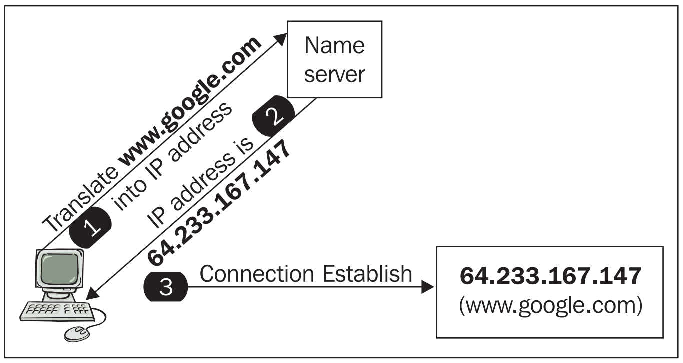

Name server
1
Translate www.google.com into IP address
2
IP address is 64.233.167.147
3
Connection Establish
64.233.167.147
(www.google.com)

Figure 1.1: It is necessary to translate a name to an IP address before establishing a connection

Just before the connection with the `www.google.com` web server is made, the `www.google.com` DNS name is translated into an IP address and only then is the connection actually established.

It is practical to use an IP address instead of a domain name whenever we suspect that the DNS on the computer is not working correctly. Although it seems unusual, in this case, we can write something like:

```
ping 64.233.167.147
http://64.233.167.147
```

or send email to

```
dostalek@[64.233.167.147]
```

However, the reaction can be unexpected, especially for the email, HTTP, and HTTPS protocols. Mail servers do not necessarily support transport to servers listed in brackets. HTTP will return to us the primary home page, and the HTTPS protocol will complain that the server name does not match the server name in the server's certificate.

1.1 Domains and Subdomains

The entire Internet is divided into domains, i.e., name groups that logically belong together. The domains specify whether the names belong to a particular company, country, and so forth. It is possible to create subgroups within a domain that are called **subdomains**. For example, it is possible to create department subdomains for a company domain. The domain name reflects a host's membership in a group and subgroup. Each group has a name affiliated with it. The domain name of a host is composed from the individual group names. For example, the host named `bob.company.com` consists of a host named `bob` inside a subdomain called `company`, which is a subdomain of the domain `com`.

The domain name consists of strings separated by dots. The name is processed from left to right. The highest competent authority is the root domain expressed by a dot (`.`) on the very right (this dot is often left out). **Top Level Domains** (**TLD**) are defined in the root domain. We have two kind of TLD, **Generic Top Level Domain** (**gTLD**) and **Country Code Top Level Domain** (**ccTLD**). Well known gTLDs are `edu`, `com`, `net`, and `mil` which are used mostly in the USA. According to ISO 3166, we also have two letter ccTLD for individual countries. For example, the `us` domain is affiliated with USA. However ccTLD are used mostly outside the USA. A detailed list of affiliated ccTLD and their details are listed in Appendix A.

The TLD domains are divided into subdomains for particular organizations, for example, `coca-cola.com`, `mcdonalds.com`, `google.com`. Generally, a company subdomain can be divided into lower levels of subdomains, for example, the company Company Ltd. can have its subdomain as `company.com` and lower levels like `bill.company.com` for its billing department, `sec.company.com` for its security department, and `head.company.com` for its headquarters.

The names create a tree structure as shown in the figure:

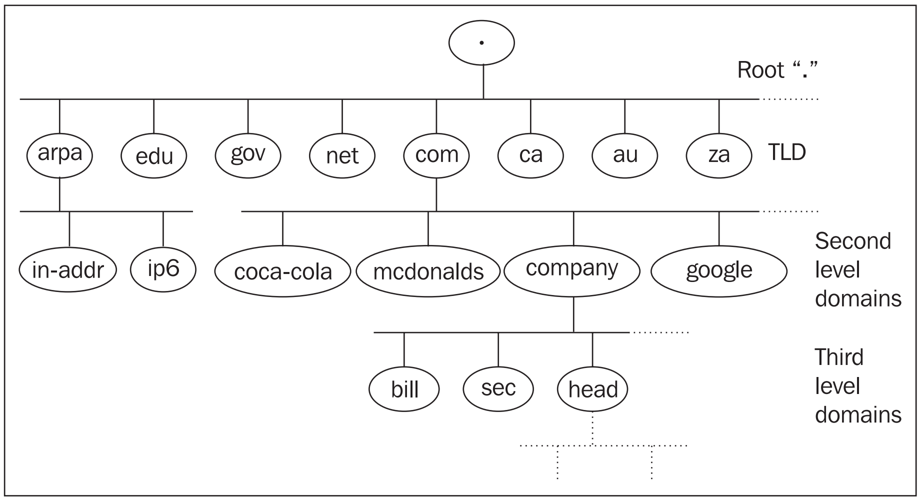


Figure 1.1a: The names in the DNS system create a tree structure

The following list contains some other registered gTLDs:

- The `.org` domain is intended to serve the noncommercial community.
- The `.aero` domain is reserved for members of the air transport industry.
- The `.biz` domain is reserved for businesses.
- The `.coop` domain is reserved for cooperative associations.
- The `.int` domain is only used for registering organizations established by international treaties between governments.
- The `.museum` domain is reserved for museums.
- The `.name` domain is reserved for individuals.
- The `.pro` domain is being established; it will be restricted to credited professionals and related entities.

1.2 Name Syntax

Names are listed in a dot notation (for example, `abc.head.company.com`). Names have the following general syntax:

```
string.string.string ………string.
```

where the first string is a computer name, followed by the name of the lowest inserted domain, then the name of a higher domain, and so on. For unambiguousness, a dot expressing the root domain is also listed at the end.

The entire name can have a maximum of 255 characters. An individual string can have a maximum of 63 characters. The string can consist of letters, numbers, and hyphens. A hyphen cannot be at the beginning or at the end of a string. There are also extensions specifying a richer repertoire of characters that can be used to create names. However, we usually avoid these additional characters because they are not supported by all applications.

Both lower and upper case letters can be used, but this is not so easy. From the point of view of saving and processing in the DNS database, lower and upper case letters are not differentiated. In other words, the name `newyork.com` will be saved in the same place in a DNS database as `NewYork.com` or `NEWYORK.com`. Therefore, when translating a name to an IP address, it does not matter whether the user enters upper or lower case letters. However, the name is saved in the database in upper and lower case letters; so if `NewYork.com` was saved in the database, then during a query, the database will return "`NewYork.com.`". The final dot is part of the name.

In some cases, the part of the name on the right can be omitted. We can almost always leave out the last part of the domain name in application programs. In databases describing domains the situation is more complicated:

- It is almost always possible to omit the last dot.
- It is usually possible to omit the end of the name, which is identical to the name of the domain, on computers inside the domain. For example, inside the `company.com` domain it is possible to just write `computer.abc` instead of `computer.abc.company.com`. (However, you cannot write a dot at the end!) The domains that the computer belongs to are directly defined by the `domain` and `search` commands in the resolver configuration file. There can be several domains of this kind defined (see Section 1.9).

1.3 Reverse Domains

We have already said that communication between hosts is based on IP addresses, not domain names. On the other hand, some applications need to find a name for an IP address—in other words, find the reverse record. This process is the translation of an IP address into a domain name, which is often called **reverse translation**.

As with domains, IP addresses also create a tree structure (see Figure 1.2). Domains created by IP addresses are often called reverse domains. The pseudodomains `inaddr-arpa` for IPv4 and `IP6.arpa` for IPv6 were created for the purpose of reverse translation. This domain name has historical origins; it is an acronym for *inverse addresses in the Arpanet*.

Under the domain `in-addr.arpa`, there are domains with the same name as the first number from the network IP address. For example, the `in-addr.arpa` domain has subdomains 0 to 255. Each of these subdomains also contains lower subdomains 0 to 255. For example, network 195.47.37.0/24 belongs to subdomain `195.in-addr.arpa`. This actual subdomain belongs to domain `47.195.in-addr.arpa`, and so forth. Note that the domains here are created like network IP addresses written backwards.

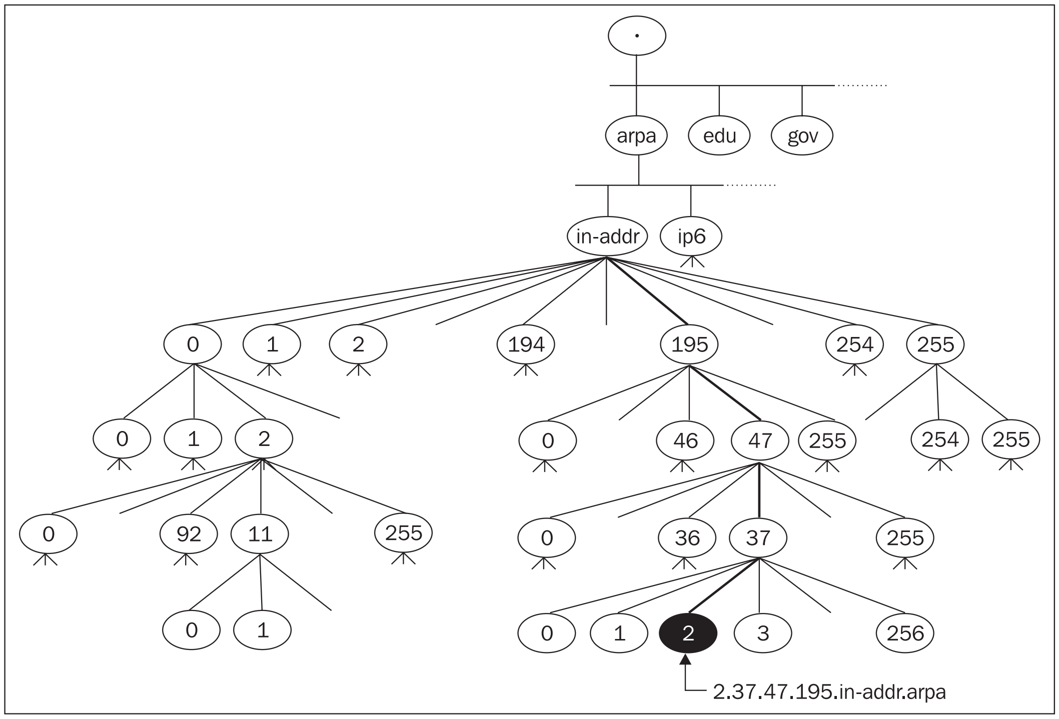


Figure 1.2: Reverse domain to IP address 195.47.37.2

This whole mechanism works if the IP addresses of classes A, B, or C are affiliated. But what should you do if you only have a subnetwork of class C affiliated? Can you even run your own name server for reverse translation? The answer is yes. Even though the IP address only has four bytes and a classic reverse domain has a maximum of three numbers (the fourth numbers are already elements of the domain—IP addresses), the reverse domains for subnets of class C are created with four numbers. For example, for subnetwork 194.149.150.16/28 we will use domain 16.150.149.194.in-addr.arpa. It is as if the IP address suddenly has five bytes! This was originally a mistake in the implementation of DNS, but later this mistake proved to be very practical so it was standardized as an RFC. We will discuss this in more detail in Chapter 7. You will learn more about reverse domains for IPv6 in Section 3.5.3.

1.4 Domain 0.0.127.in-addr.arpa

The IP address 127.0.0.1 presents an interesting complication. Network 127 is reserved for loopback, i.e., a software loop on each computer. While other IP addresses are unambiguous within the Internet, the address 127.0.0.1 occurs on every computer. Each name server is not only an authority for common domains, but also an authority (primary name server) to domain `0.0.127.in-addr.arpa`. We will consider this as given and will not list it in the chart, but be careful not to forget about it. For example, even a caching-only server is a primary server for this domain. Windows 2000 pretends to be the only exception to this rule, but it would not hurt for even Windows 2000 to establish a name server for zone `0.0.127.in-addr.arpa`.

1.5 Zone

We often come across the questions: What is a zone? What is the relation between a domain and a zone? Let us explain the relationship of these terms using the `company.com` domain.

As we have already said, a domain is a group of computers that share a common right side of their domain name. For example, a domain is a group of computers whose names end with `company.com`. However, the domain `company.com` is large. It is further divided into the subdomains `bill.company.com`, `sec.company.com`, `sales.company.com`, `xyz.company.com`, etc. We can administer the entire `company.com` domain on one name server, or we can create independent name servers for some subdomains. (In Figure 1.3, we have created subordinate name servers for the subdomains `bill.company.com` and `head.company.com.`) The original name server serves the domain `company.com` and the subdomains `sec.company.com`, `sales.company.com`, and `xyz.company.com`—in other words, the original name server administers the `company.com` zone. The zone is a part of the domain namespace that is administered by a particular name server.

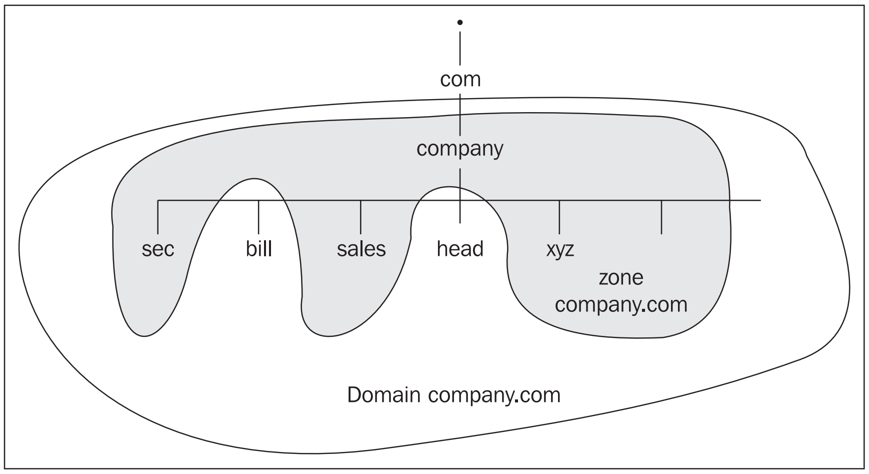


Figure 1.3: Zone company.com

A zone containing data of a lower-level domain is usually called a **subordinate zone**.

1.5.1 Special Zones

Besides classic zones, which contain data about parts of the domains or subdomains, special zones are also used for DNS implementation. Specifically, the following zones are used:

- **Zone stub**: Zone stub is actually a subordinate zone that only contains information about what name servers administer in a particular subdomain (they contain the NS records for the zone). The zone stub therefore does not contain the entire zone.
- **Zone cache/hint**: A zone hint contains a list of root name servers (non-authoritative data read into memory during the start of the name server). Only BIND version 8 and later use the name hint for this type of zone. In previous versions, a name cache zone was used. Remember that the root name servers are an authority for a root domain marked as a dot (`.`).

1.6 Reserved Domains and Pseudodomains

It was later decided that other domains could also be used as TLD. Some TLD were reserved in RFC 2606:

- The `test` domain for testing
- The `example` domain for creating documentation and examples
- The `invalid` domain for evoking error states
- The `localhost` domain for software loops

Domains that are not directly connected to the Internet can also exist, i.e., computers that do not even use the TCP/IP network protocol therefore do not have an IP address. These domains are sometimes called **pseudodomains**. They are meaningful especially for electronic mail. It is possible to send an email into other networks and then into the Internet with the help of a pseudodomain (like DECnet or MS Exchange).

In its internal network, a company can first use TCP/IP and then DECnet protocol. A user using TCP/IP in the internal network (for example, `Daniel@computer.company.com`) is addressed from the Internet. But how do you address a user on computers working in the DECnet protocol?

To solve this, we insert the fictive `dnet` pseudodomain into the address. The user Daniel is therefore addressed `Daniel@computer.dnet.company.com`. With the help of DNS, the entire email that was addressed into the `dnet.company.com` domain is redirected to a gateway in DECnet protocol (the gateway of the `company.com` domain), which performs the transformation from TCP/IP (for SMTP) into DECnet (for Mail-11).

1.7 Queries (Translations)

Most common queries are translation of a hostname to an IP address. It is also possible to request additional information from DNS. Queries are mediated by a resolver. The **resolver** is a DNS client that asks the name server. Because the database is distributed worldwide, the nearest name server does not need to know the final response and can ask other name servers for help. The name server, as an answer to the resolver, then returns the acquired translation or returns a negative answer. All communication consists of queries and answers.

The name server searches in its cache memory for the data for the zone it administers during its start. The primary name server reads data from the local disk; the secondary name server acquires data from the primary name server by a query zone transfer of the administered zones and also saves them into the cache memory. The data stored within the primary and secondary name servers is called **authoritative data**. Furthermore, the name server reads from its memory cache/hint the zone data, which is not part of the data from its administered zone (local disk), but nonetheless enables this data to connect with the root name servers. This data is called **nonauthoritative data**. In the terminology of BIND program version 8 and 9, we sometimes do not speak of them as primary and secondary servers, but as master servers and slave servers.

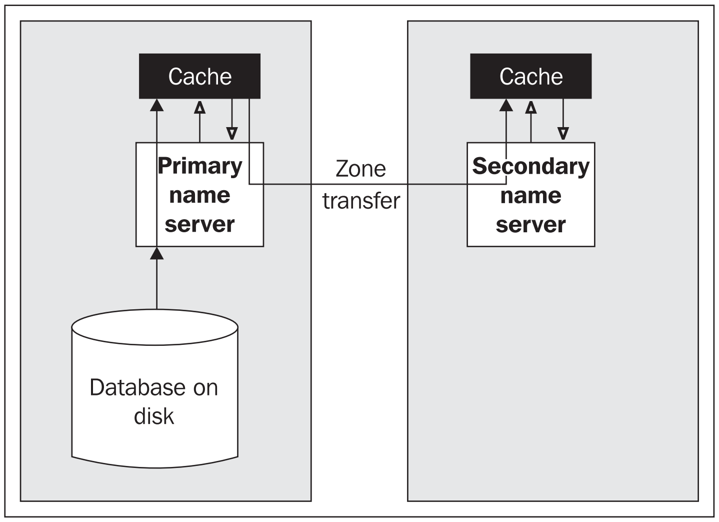


Figure 1.4: Primary name server loads data from a disk, while the secondary server acquires data by *zone transfer* query

Name servers save into their cache memory positive (and sometimes even negative) answers to queries that other name servers have to ask for. From the point of view of our name server, this data acquired from other name servers is also non-authoritative, thereby saving time when processing repeated queries.

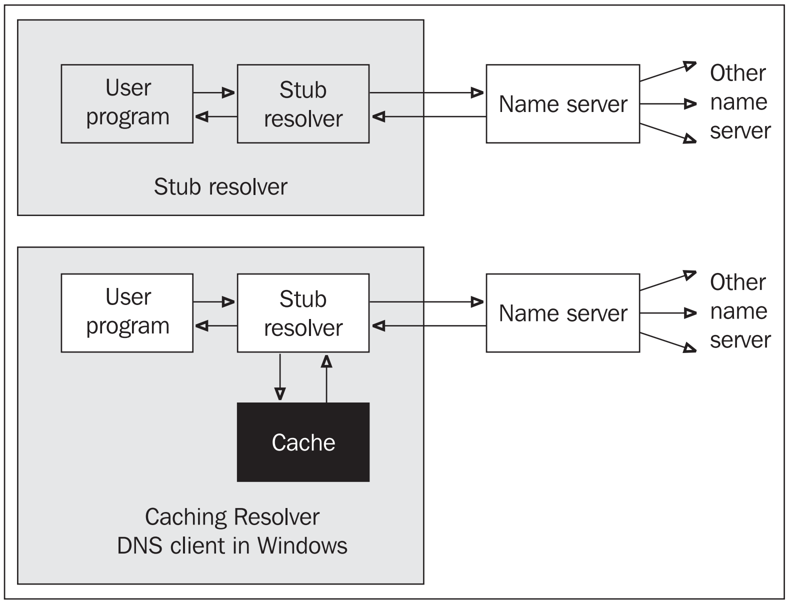


Figure 1.5: Stub resolvers and caching resolvers

Requirements for translations occur in a user program. The user program asks a component within the operating system, which is called a **resolver**, for a translation. The resolver transfers the query for translation to a name server. In smaller systems, there is usually only a stub resolver. In such cases, the resolver transfers all requirements by DNS protocol to a name server running on another computer (see Figure 1.5). A resolver without cache memory is called a *stub resolver*. It is possible to establish cache memory for a resolver even in Windows 2000, Windows XP, etc. This service in Windows is called **DNS Client**. (I think this is a little bit misleading as a stub resolver is not a *proper* DNS client!)

Some computers run only a resolver (either stub or caching); others run both a resolver and a name server. Nowadays, a wide range of combinations are possible (see Figure 1.6) but the principle remains the same:

1. The user inserts a command, then the hostname needs to be translated into an IP address (in Figure 1.6, number 1).
2. If the resolver has its own cache, it will attempt to find the result within it directly (2).
3. If the answer is not found in the resolver cache (or it is a stub), the resolver transfers the request to a name server (3).
4. The name server will look for the answer in its cache memory.
5. If the name server does not find the answer in its cache memory, it looks for help from other name servers.
6. The name server can contact more name servers by a process referred to as iteration. By iteration, the name server can access or contact a name server, which is an authority on the answer. The authoritative name server will then give a last resort answer (negatively if there is no information in DNS corresponding with the inserted name).
7. But if the process described above does not return the result fast enough, the resolver repeats its query. If there are more name servers listed in the resolver configuration, then it will send the next query to the next name server listed in the directory (i.e., another name server). The directory of name servers is processed cyclically. The cycle starts for the particular query from the name server, which is listed in the first position.

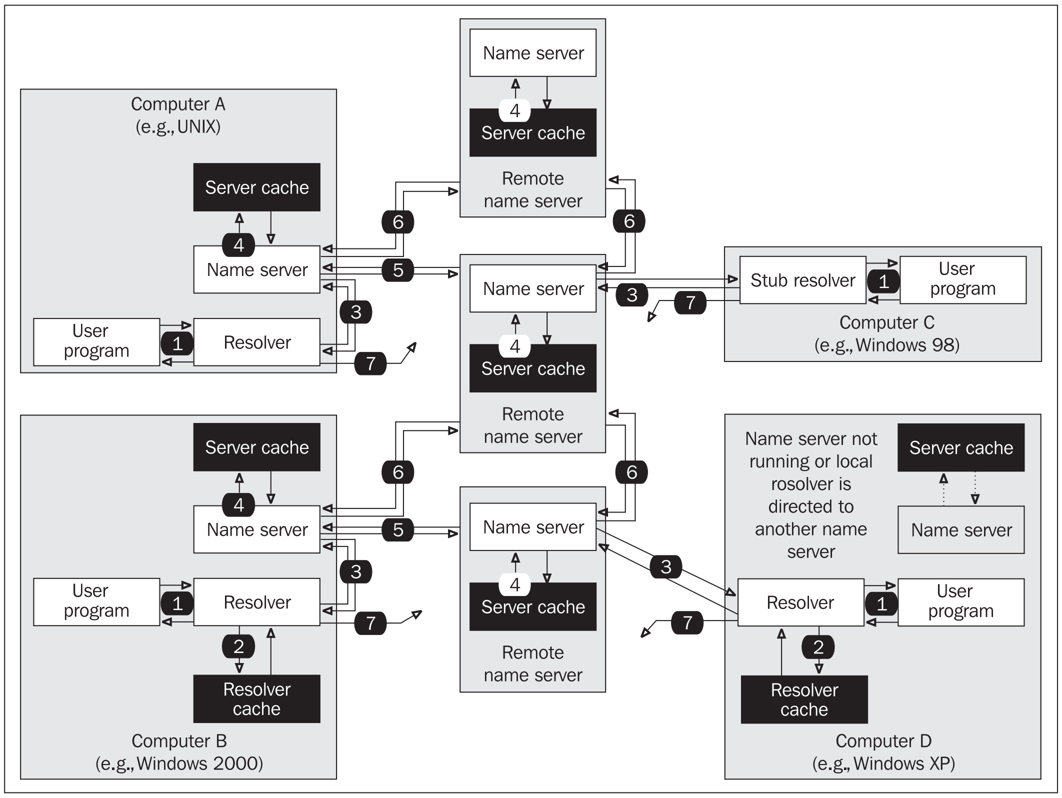


Figure 1.6: Name server and resolver

DNS uses both UDP and TCP protocols for the transport of its queries/answers. It uses port 53 for both protocols (i.e., ports 53/UDP and 53/TCP). Common queries such as the translation of a name to an IP address and vice versa are performed by UDP protocol. The length of data transported by UDP protocol is implicitly limited to 512 B (a truncation flag can be used to signal that the answer did not fit into 512 B and it is therefore necessary for the query answer to be repeated by the TCP protocol). The length of UDP packets is limited to 512 B because a fragmentation could occur for larger IP datagrams. DNS does not consider fragmentation of UDP as sensible. Queries transporting zone transfer data occur between the primary and secondary name servers and are transported by TCP protocol.

Common queries (such as the translation of a name to an IP address and vice versa) are performed with the help of datagrams in UDP protocol. The translations are required by a client (resolver) on the name server. If the name server does not know what to do, it can ask for translation (help) from other name servers. Name servers solve questions among themselves by iteration, which always starts from the root name server. More details are available in Section 1.10.

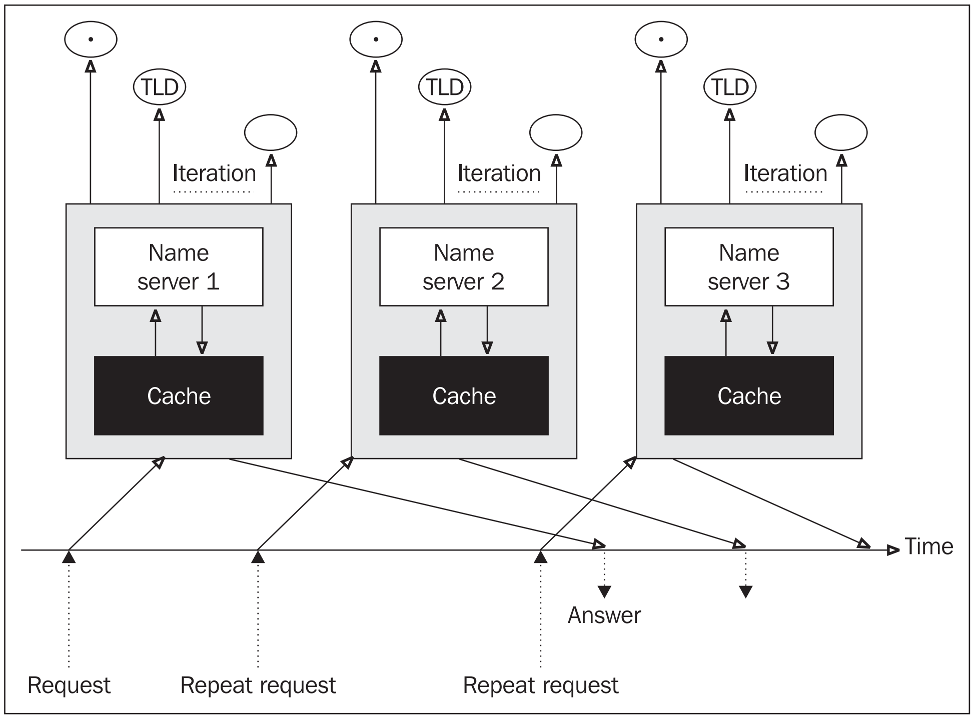


Figure 1.7: Required answer for a translation

There is a rule in the Internet that a database with data needed for translations is always saved on at least two independent computers (independent name servers). If one is unavailable, the translation can be performed by the other computer.

In general, we cannot expect that all name servers are accessible all the time. If the TCP protocol is used for a translation, attempts to establish a connection with an inaccessible name server would cause long time intervals while the TCP protocol is trying to connect. Only when this time interval is over is it possible to connect to the next name server.

The solution for this in UDP protocol is more elegant: A datagram containing a request for the translation is sent to the first server. If the answer does not come back within a short time-out interval, then a datagram with a request is sent to another name server, if the answer does not come back again, it is sent to the next one, and so on. If all possible name servers are used, it will start again from the first one, and the whole cycle repeats until the answer comes back or the set interval times out.

1.7.1 Round Robin

Round Robin is a technique that can be used to equally load several machines (Load Balancing). It is possible to use this technique for the majority of name servers (including Windows 2000/2003). This is a situation where we have more than one IP address for one name in DNS. For example, we may operate an exposed web server and because the performance of the machine is not sufficient, we buy another or two more. We start running the web server on all three of them (for example, `www.company.com`). The first one has an IP address 195.1.1.1, the second one 195.1.1.2, and the third one 195.1.1.3. There will be three records in DNS for `www.company.com`, and each of them will have a different IP address. Round Robin technique ensures that the answer to the:

1. first query (to the first user) will be that the web server return addresses 195.1.1.1, 195.1.1.2, and 195.1.1.3
2. the answer to the next query (to the second user) will be that the server return IP addresses 195.1.1.2, 195.1.1.3, and 195.1.1.1.
3. the answer to te next query (may be 3rd user) will return IP addresses 195.1.1.3, 195.1.1.1, and 195.1.1.2.
4. procedure are repeating from 1st point again and again.

1.8 Resolvers

A **resolver** is a component of the system dealing with the translation of an IP address. A resolver is a client; it is not a particular program. It is a set of library functions that are linked with application programs requiring services such as Telnet, FTP, browsers, and so on. For example, if Telnet needs to translate the name of a computer to its IP address, it calls the particular library functions.

The client (in this case, the aforementioned Telnet) calls the library functions (`gethostbyname`), which will formulate the query, and send it to the name server.

Time limitations must also be considered. It is possible that a resolver does not receive an answer to its first query, while the next one with the same content is answered correctly (while the server is waiting for the first query, it manages to obtain the answer for the second query from another name server, so the first query was not answered, because the response of its name server took too long). From the user's point of view, it seems that the translation was not managed on the first try, but was completed by processing it again. The use of the UDP protocol causes a similar effect. Note that it can also happen that the server did not receive the request for the translation at all, because the network is overloaded, and the UDP datagram has been lost somewhere along the way.

1.8.1 Resolver Configuration in UNIX

The configuration file for a resolver in the UNIX operating system is `/etc/resolv/conf`. It usually contains two types of lines (the second command can be repeated several times):

```
domain          the name of the local domain
nameserver      IP address of name server
```

If the user inserted the name without a dot at the end, the resolver will add the domain name from the `domain` command after the inserted name, and will try to transfer it to the name server for translation. If the translation is not performed (a negative answer has been received from the name server), the resolver will try to translate the actual name without the suffix from the `domain` command.

Some resolvers enable the `search` command. This command allows us to specify more names of local domains.

The IP address of a name server that the resolver should contact is specified by the `nameserver` command. It is recommended to use more `nameserver` commands for times when some name server is not available.

> The IP address of a name server always has to be stated in the configuration file of the resolver, not the domain name of the name server!

When configuring the resolver and name server on the same machine, the `nameserver` command can be directed to a local name server 127.0.0.1 (but this is not necessary).

Other parameters of the resolver (for example, the maximum number of `nameserver` commands) can be set in the configuration file of the operating system kernel. This file is often called `/usr/include/resolv.h`. Afterwards, of course, a new compilation of the kernel operating system must follow.

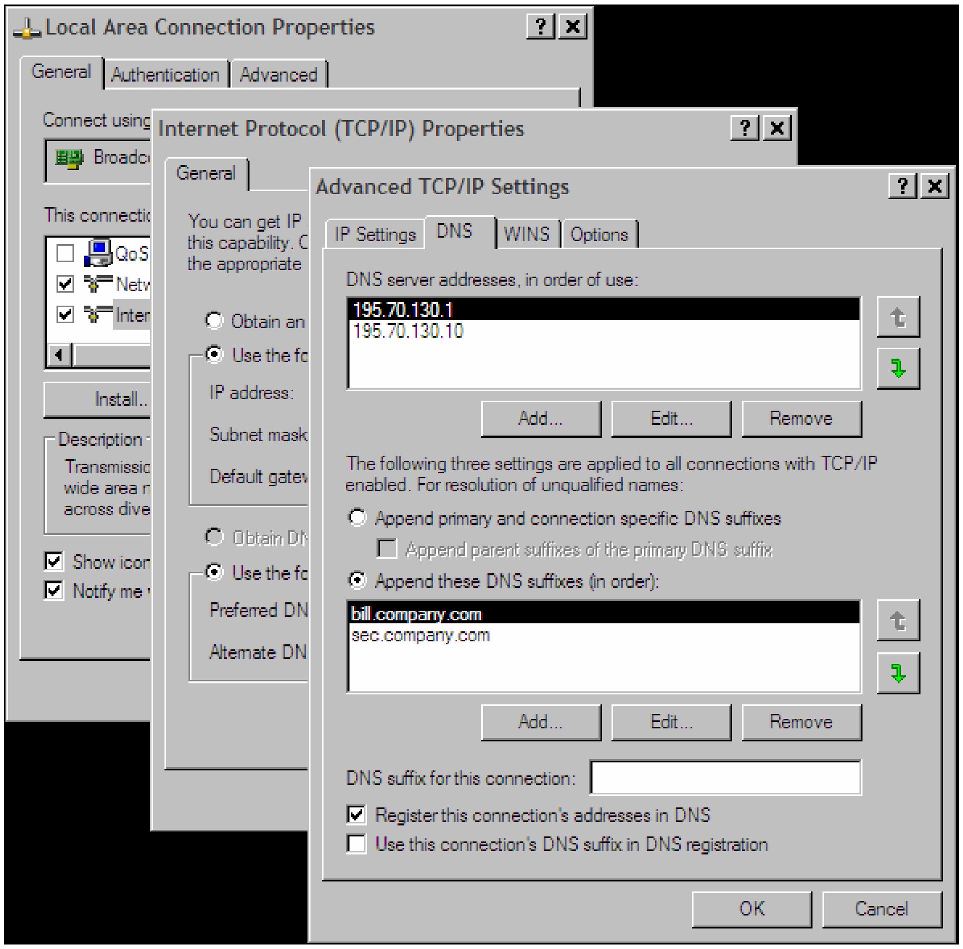
Generally, it is also possible to configure all computers without the use of DNS. Then all requests for address translations are performed locally with the help of the `/etc/hosts` file (in Windows `%System_Root%/System32/Drivers/etc/hosts`). It is possible to combine both methods (the most typical variant); however, we need to be careful about the content of the database `/etc/hosts`. Usually it is also possible to set the order in which the databases are supposed to be browsed. Usually one `/etc/hosts` file is browsed and afterwards the DNS.

1.8.2 Resolver Configuration in Windows

There is an interesting situation in Windows 2000 and higher. Here we still have the previously mentioned DNS Client service. It is an implementation of a caching resolver. This service is started implicitly. It is strictly recommended in the documentation not to stop this service. However, according to my tests, Windows acts like a station with a stub resolver after stopping this service.

The content of a resolver cache can even be written out by a `ipconfig /displayDNS` command or deleted by `ipconfig /flushDNS` command.

The content of a `%System Root%/System32/Drivers/etc/hosts` file whose content is not changed by the `ipconfig /flushDNS` command is also a part of the cache resolver. The cache resolver can be parameterized by the insertion or change of keys in the Windows register folder `HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Dnscache/Parameters`, for example, by a `NegativeCacheTime` key, where you can specify a time period within which negative answers kept in the cache resolver can be changed.



DNS server addresses, in order of use:

195.70.130.1
195.70.130.10

The following three settings are applied to all connections with TCP/IP enabled. For resolution of unqualified names:

( ) Append primary and connection specific DNS suffixes

[ ] Append parent suffixes of the primary DNS suffix

(•) Append these DNS suffixes (in order):

bill.company.com
sec.company.com

DNS suffix for this connection:

[✓] Register this connection's addresses in DNS

[ ] Use this connection's DNS suffix in DNS registration

Figure 1.8: Configuration of a resolver in Windows XP

In older Windows versions, the configuration of a resolver was as simple as it was in UNIX. The difference was only in the fact that a text configuration file was not created by a text editor, but the values were inserted into a particular window. With Windows XP this particular configuration window of the resolver (Figure 1.8) contains a lot more information.

It is necessary to look at Windows XP and higher from a historical point of view. The LAN Manager System based on NetBIOS protocol was the predecessor of the Windows network. NetBIOS protocol also uses names of computers, which it needs to translate to network addresses in the network layer. When Windows uses TCP/IP as a network protocol, it needs to translate the names of computers to IP addresses and vice versa.

LAN Manager implemented its own system of names. Names with particular IP addresses were saved locally in a `%SystemRoot%/System 32/Drivers/etc/lmhosts` file. Later Windows implemented a DNS analogy, a database called WINS (Windows Internet Names Service).

The translation of names is an interesting problem in Windows. When a translation is not found either in an `lmhosts` file or on WINS server, it is then sent to a broadcast requesting whether the searched for computer is present on the LAN. Searching in DNS after the implementation of DNS into Windows has extended the entire mechanism. So programs in Windows 2000, which have LAN Manager system as a precursor search for the translation:

1. In the LAN Manager cache of a local computer (`nbtstat -c` command lists the cache). It is a cache of the NetBIOS protocol. Rows of the `lmhosts` file, having the `#PRE` string as a last parameter, are loaded into this cache when a computer starts. If the `lmhosts` file is changed, we can force reloading of these rows into a cache by the `nbtstat -R` command.
2. On WINS servers. By a broadcast or multicast on LAN.
3. In the `lmhosts` file.
4. In a resolver cache (even the content of hosts file is read into it).
5. On DNS servers.

And programs (for example, the `ping` command) that are Internet oriented search for the translation:

1. In the resolver cache (even the content of hosts file is read into it).
2. On DNS Servers.
3. On WINS servers.
4. By a broadcast or multicast packet of NetBIOS protocol.
5. In the `lmhosts` file.

So if you make a mistake in the name of the computer in the `ping` command, then in the record of a MS Network Monitor program or in the record of an Ethereal program (visit `http://www.ethereal.com` for additional information) you will also be able to see the packets of NetBIOS protocol and even the search conducted by a broadcast.

Now to the configuration of a resolver in Windows XP in Figure 1.8. First we will insert the IP addresses of name servers into the upper window (DNS server's address, in order of use). It is not necessary to insert them if we get them during the start up of the computer, for example, from a DHCP server or during the establishment of a dial-up connection with the help of PPP protocol.

Furthermore, there are two options here:

1. Select Append primary and connection specific in DNS suffixes in the DNS tab (this option is not selected in Figure 1.8); the translation is performed as follows:
    - If the required name contains a dot, then the resolver tries to translate the name without adding a suffix.
    - If the name does not contain a dot, it tries to translate the inserted name after which it has added a dot and a domain name of a Windows domain (configured on Properties in the Computer Name tab).
    - It tries to translate the inserted name after which it has added a dot and a name of a chain in a field DNS suffix for this connection.

2. Click Append these DNS suffixes (in order); the translation is performed as follows:
    - If the required name contains a dot then the resolver tries to translate the name without adding a suffix.
    - It tries to add particular suffixes according to a list listed in the window below the mentioned option.

So if you make a mistake in the name of the computer and hit a nonexistent name xxx, then because you have selected a second option, the resolver will first try to translate the name `xxx.bill.company.com` and then a name `xxx.sec.company.com`. In both cases, it will generate a query to the name server 195.70.130.1 for each of these translations and then if you do not receive the answer in time, it will repeat the question to the server 195.70.130.10, and the whole cycle is repeated until the time limit is exceeded.

1.9 Name Server

A name server keeps information for the translation of computer names to IP addresses (even for reverse translations). The name server takes care of a certain part from the space of names of all computers. This part is called the zone (at minimum it takes care of zone 0.0.127.in-addr.arpa).

A domain or its part creates the zone. The name server can with the help of an NS type record (in its configuration) delegate administration of a subdomain to a subordinate name server.

The name server is a program that performs the translation at the request of a resolver or another name server. In UNIX, the name server is materialized by the named program. Also the name **BIND** (**Berkeley Internet Name Domain**) is used for this name server.

Types of name servers differ according to the way in which they save data:

- **Primary name server/primary master** is the main data source for the zone. It is the authoritative server for the zone. This server acquires data about its zone from databases saved on a local disk. Names of these types of servers depend on the version of BIND they use. While only the primary name server was used for version 4.x, a primary name master is used for version 8. The administrator manually creates databases for this server. The primary server must be published as an authoritative name server for the domain in the SOA resource record, while the primary master server does not need to be published. There is only one of this type of server for each zone.
- **Master name server** is an authoritative server for the zone. The master server is always published as an authoritative server for the domain in NS records. The master sever is a source of data of a zone for the subordinate servers (slave/secondary servers). There can be several master servers. This type of server is used for Bind version 8 and later.
- **Secondary name server/slave name server** acquires data about the zone by copying the data from the primary name server (respectively from the master server) at regular time intervals. It makes no sense to edit these databases on the secondary

name servers, although they are saved on the local server disk because they will be rewritten during further copying. This type of name server is also an authority for its zones, i.e., its data for the particular zone is considered irrevocable (authoritative). The name of this type of server depends again on the version of BIND it uses. For version 4, only the secondary name was used, the term slave server was used for a completely different type of server. In version 8 you can come across both names.

- **Caching-only name server** is neither a primary nor secondary name server (it is not an authority) for any zone. However, it uses the general characteristics of name servers, i.e., it saves data that comes through its cache. This data is called nonauthoritative. Each server is a caching server, but by the words caching, we understand that it is neither a primary nor secondary name server for any zone. (Of course, even a caching-only server is a primary name server for zone 0.0.127.in-addr.arpa, but that does not count).
- **Root name server** is an authoritative name server for the root domain (for the dot). Each root name server is a primary server, which differentiates it from other name servers.
- **Slave name server** (in BIND version 4 terminology) transmits questions for a translation to other name servers; it does not perform any iteration itself.
- **Stealth name server** is a secret server. This type of name server is not published anywhere. It is only known to the servers that have its IP address statically listed in their configuration. It is an authoritative server. It acquires the data for the zone with the help of a zone transfer. It can be the main server for the zone. Stealth servers can be used as a local backup if the local servers are unavailable.

The architecture of a master/slave system is shown in the following figure:

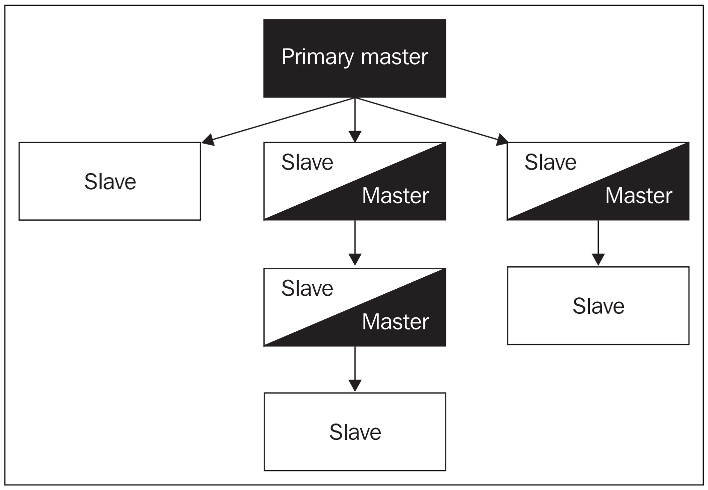


Figure 1.9: Master/slave architecture

One name server can be a master (primary) server for one zone and a slave (secondary) server for another.

From the point of view of a client, there is no difference between master (primary) and slave (secondary) name servers. Both contain data of similar importance—both are authoritative for the particular zone. The client does not even need to know which server is the master (primary server) and which one is the slave (secondary). On the other hand, a caching server is not an authority, i.e., if it is not able to perform the translation, it contacts the authoritative server for the particular zone.

So if the hostmaster change some information on the master server (i.e. adds another computer name into the database), then the databases on all slave servers are automatically corrected after a time set by a parameter in the SOA resource record (if the hostmaster only corrected the database manually on a secondary name server, the correction would disappear at the same time!). A problem occurs when the user receives the first answer from the slave server at a time when the slave server has not been updated. The answer is negative, i.e., such a computer is not in the database.

Even worse is the following case: the master server operates correctly, but there is no data for the zone on the slave server because zone transfer failed. The clients receive authoritative answers from the master server or the slave server by chance. When the client receives an answer from the master server, the answer is correct. When the client receives an answer from the slave server, the answer is negative. But the user doesn't know which server is correct and which is wrong . Then the user says, "*First I receive a response to my query and second time I do not*."

Authoritative data comes from the database which is stored on the primary master's disk. Nonauthoritative data comes from other nameservers ("from the network"). There is only one exception. The name server needs to know the root name servers to ensure proper functioning of the name server. However, it is not an authority for them usually, still each name server has own nonauthoritative information about root servers on the disk. It is implemented by a cache command in BIND version 4 or zone cache/hint in BIND version 8 and later.

The iteration process of a translation of the name `abc.company.com` to an IP address is shown in the following figure below:

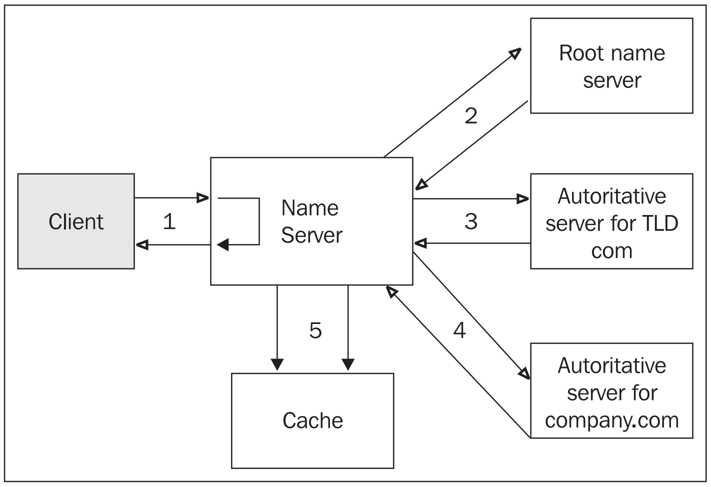

Root name server
2
Client
1
Name Server
3
Autoritative server for TLD com
5
4
Autoritative server for company.com
Cache

Figure 1.10: Translation of a domain name abc.company.com to IP address

The step-by-step process is as follows:

1. The resolver formulates the requirement to the name server and expects an unambiguous answer. If the name server is able to answer, it sends the answer immediately. It searches for the answer in its cache memory (5). Authoritative data from the disk databases is acquired as well as nonauthoritative data acquired during previous translations. If the server does not find the answer in its cache memory, it contacts other servers. It always begins with a root name server. If the name server does not know the answer itself, it contacts the root name server. That is why each name server must know the IP addresses of root name servers. If no root name server is available (as is, for example, the case for all closed Intranets), then after several unsuccessful attempts, the entire translation process collapses.
2. The root name server finds out that the information about the `.com` domain was delegated by NS resource record to the subordinate name server and it will return this subordinate name server's IP addresses (IP address of authoritative name servers for the zone `.com`).
3. Our name server turns to the authoritative server for the `.com` domain and finds out that the information about the `company.com` domain was delegated by NS type resource record to the subordinate name server and will return this subordinate name server's IP addresses (IP address of authoritative name servers for `company.com` zone).
4. Our name server then turns to the authoritative name server for the `company.com` domain, which will solve its query (or not). The answer from authoritative name server for relevant zone is marked as an authoritative answer. The result is transmitted to the client (1).
5. The information, which the server has gradually received, will also be saved into the cache. The answer to the next similar question is looked up in cache and returned directly from cache. But this next answer is not marked as authoritative.

The name server even saves answers into the cache memory described in the previous five points (translation of `abc.company.com`). It can then use the answers from the cache for the following translations to save time, but it also helps the root name servers. However, if you require the translation of a name from TLD that is not in the cache, then the root name server is really contacted. From this we can see that the root servers in the Internet will be heavily burdened and their unavailability would damage communication on the entire Internet.

The name server does not require the complete (recursive) answer. Important name servers (for example, root name servers or TLD name servers) do not even have to produce recursive answers, and hence avoid overloading themselves and restricting their availability. It is not possible to direct the resolver of your computer to them.

The `nslookup` program is a useful program for the administrator of the name server. If you want to perform questions on a name server with the `nslookup` program, then forbid iteration (recursive questions) and the addition of domain names from the configuration file of the resolver with the commands:

```
$ nslookup
 set norecurse
 set nosearch
```

1.10 Forwarder Servers

There is another type of server, called a forwarder server. The characteristics of this server are not connected with whether it is a primary or secondary server for any zone, but with the way in which the translation of DNS questions is performed.

So far we have said that the resolver transfers the request for the translation to a name server, i.e., it sends a query to a name server and waits for the final answer (the client sends a recursive query and waits for a final answer). If the name server is not able to answer itself, it performs a recursive translation via non-recursive queries. First it contacts the root name server. The root name server tells the resolver which name servers it must ask for answers to its query. Then it contacts the recommended name server. This name server sends many packets into the Internet.

If a company network is connected to the Internet by a slow line, then the name server loads the line by its translations. In such a case, it is advantageous to configure some of the name servers as forwarder servers.

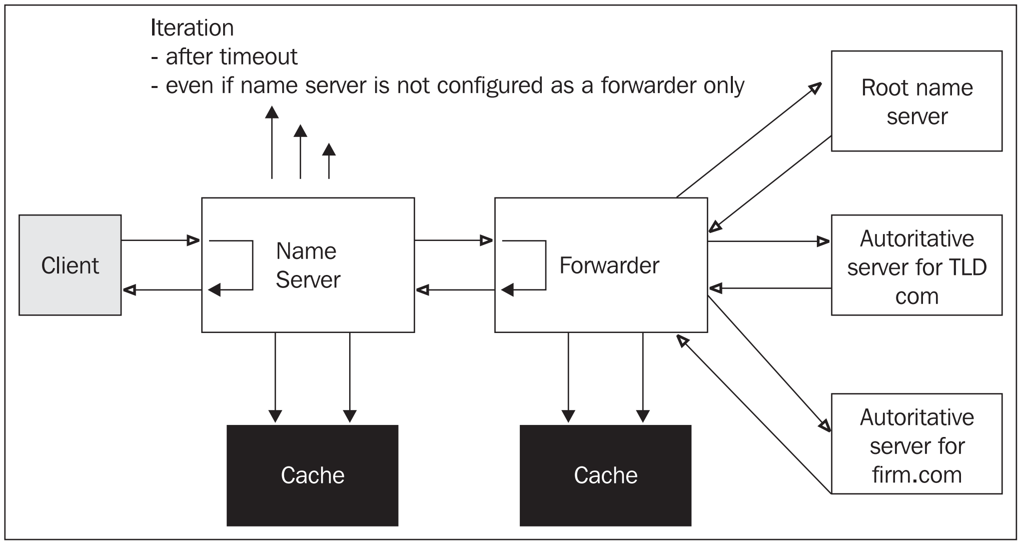

Iteration
- after timeout
- even if name server is not configured as a forwarder only
Root name server
Client
Name Server
Forwarder
Autoritative server for TLD com
Autoritative server for firm.com
Cache
Cache

Figure 1.11: Communication of a local name server with a forwarder server

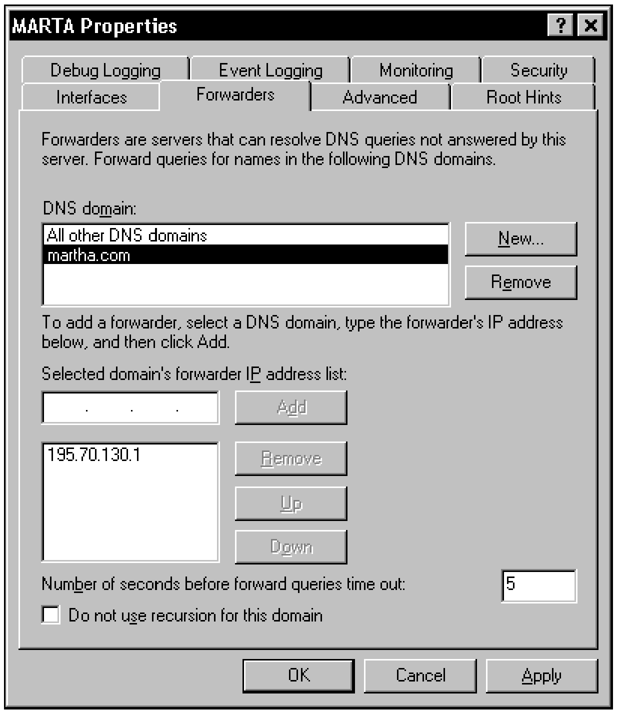
The local name server transmits the queries to the forwarder server. However, the local name server marks these queries as recursive. The forwarder server takes the request from the local name server and performs translation via non-recursive queries on the Internet by itself. It then returns only the final result to our name server.

The local name server waits for the answer from the forwarder server for the final result. If the local name server does not get the answer in the set time out limit, then it contacts the root name servers and tries to solve the case by iteration.

If the local name server is not supposed to contact the root name servers, but is supposed to only wait for the answer, then it is necessary to indicate such a server in its configuration as a *forwarder-only*. In BIND version 4.x such a server is called *slave*. Forwarder-only (slave) servers are used on intranets (behind the firewall) where contact with root name servers is not possible. The forwarder server then contacts a name server, which is part of the firewall.

The forwarder server can work as a caching-only server in both variants, and it can also be the primary or secondary name server for some zones.

It is also possible to configure forwarder servers in Windows 2003 Server as shown in the figure below:


MARTA Properties

Forwarders are servers that can resolve DNS queries not answered by this server. Forward queries for names in the following DNS domains.

DNS domain:
All other DNS domains
martha.com

To add a forwarder, select a DNS domain, type the forwarder's IP address below, and then click Add.

Selected domain's forwarder IP address list:
195.70.130.1

Number of seconds before forward queries time out: 5

☐ Do not use recursion for this domain

Figure 1.12: Forwarders configuration in Windows 2003

Run the DNS from the Administrative Tools. Right-click to your DNS server and choose Properties. Select the Forwarders tab. Click New and enter the name of the domain you want to resolve by forwarders. Insert the IP addresses of the forwarder servers below. You can insert into the Number of seconds before forward queries time out box a time limit during which the server waits for an answer from a forwarder server. We can establish a slave server by clicking the Do not use recursion for this domain option.

Anschnitt aus, Quelle:
DNS in Action
A detailed and practical guide to DNS implementation, configuration, and administration

First published: March 2006

Production Reference: 1240206

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 1-904811-78-7

`www.packtpub.com`

Cover Design by `www.visionwt.com`

This is an authorized and updated translation from the Czech language.