Jose L. Muñoz, Juanjo Alins, Oscar Esparza, Jorge Mata

Domain Name Service (DNS)

Transport Control i Gestió a Internet (TCGI)
Universitat Politècnica de Catalunya (UPC)
Dp. Enginyeria Telemàtica (ENTEL)

Contents

Chapter 1

DNS

1.1 Introduction

In an IP network, we must know the IP address of the destination device to send it an IP datagram. However, human beings are not good remembering numbers. For human beings, names are much easier to remember than numeric addresses. Like the address book of your phone links names of people with phone numbers, the Domain Name System (DNS) links names with IP addresses in the Internet. For example, the name www.upc.edu corresponds to the IP address 147.83.2.135. Thus, you can type in your browser the name www.upc.edu. instead of the IP address 147.83.2.135 and you will get the same results. A particularity of the DNS is that the translation between IP addresses and domain names may not be unique. On one hand, an IP address can translate to several names and on the other hand, a name can translate to multiple IP addresses.

1.1.1 Local Translation

Now, the question is how to implement the DNS. The simplest way is to use a local file in each computer. In Unix-like systems, we use the file /etc/hosts. This file contains lines of text with name translations in the form “address”, “long name” and “short name”. For example:

```
147.83.2.135        www.upc.edu     www
```

1.1.2 Centralized Server

At the beginning of the Internet, the local file was a correct solution because the network was a community of few hosts, small and static. However, the current Internet has a huge number of hosts (see Figure 1.1). In this context, enforcing users to populate their own “hosts” file is cumbersome and impractical.

A much better solution is to store the translations in a server and only use the “hosts” file for specific local translations. In the early days of Internet, a single server with a plain text file like the “hosts” file was used. The file and the server were maintained centrally by an organization called InterNIC.

However, as the local file solution, the centralized approach is neither a good solution to manage the DNS traffic of the current Internet (which is huge). A centralized system leads to file update and distribution problems. In addition, name collisions, in which two hosts try to get the same name are difficult to prevent in this case. Finally, maintaining consistency of the file is an arduous task as new names are added. In this context, it is clear that a decentralized solution for the DNS is necessary. Furthermore, some mechanism for delegation is also necessary so that each organization can define their translations without having to contact InterNIC each time.

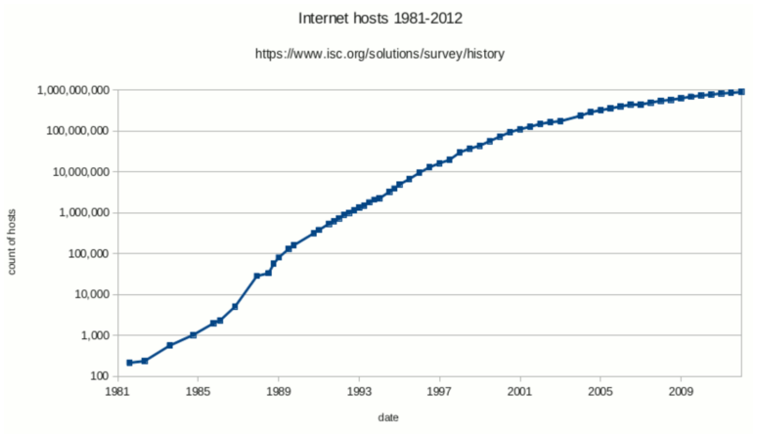


Figure 1.1: Internet Hosts.

1.1.3 Decentralized Solution

A more suitable solution for providing **efficient name translations** and **delegation** is to use a **distributed database** (multiple servers) with a **variable depth hierarchical name space** (see Figure 1.2). This is the actual solution in the current Internet.

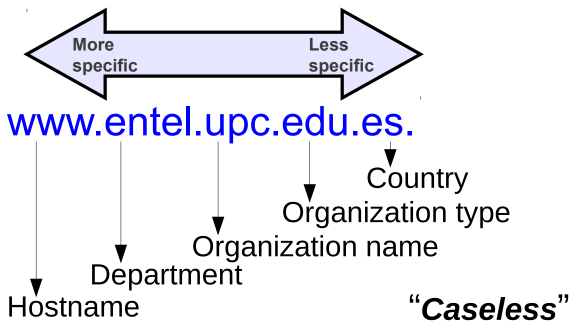


Figure 1.2: Variable Depth Hierarchical Name Space.

The variable depth hierarchical name space literally means that names can have different lengths and that they are ordered in an hierarchy. This hierarchy allows name delegation and name uniqueness. With this solution, we can have several devices with the same short name or **unqualified name**, for example www; but each device can have a different long name or **Fully Qualified Domain Name (FQDN)**. For example, the names www.upc.edu. and www.entel.upc.edu have the same unqualified name, which is www, but different FQDN names (see Figure 1.3).

1.2 Domains & Zones

1.2.1 Domains

The name space of the DNS tree is divided into **domains**. A domain includes all the names ending with a domain suffix. For example, the domain upc.edu includes all the names ending with upc.edu. It is remarkable that domains overlap. For example, the domain upc.edu includes the subdomain entel.upc.edu. Domains are classified according to their level or depth inside the DNS hierarchy. **First level domains** are managed by governmental organizations, countries or special agencies related with Internet. Examples of first level domains are:

- **edu** Universities

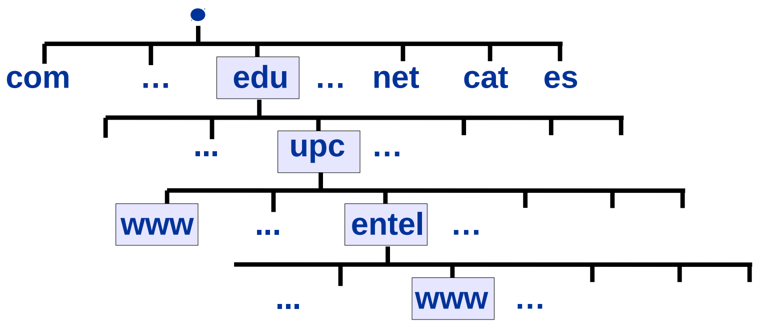


Figure 1.3: Uniqueness and Delegation.

- **com** Commercial Organizations
- **org** Non Commercial Organizations
- **net** Administrative networks
- **mil** US Army
- **gov** US Government

Second level domains are managed by private entities. An example of second level domain is "upc.edu.“ Additionally, there is a top level or root domain. This domain is noted as dot "." and it is managed by InterNIC (see Figure 1.3). For example, let us consider the simplified DNS tree of Figure 1.4.

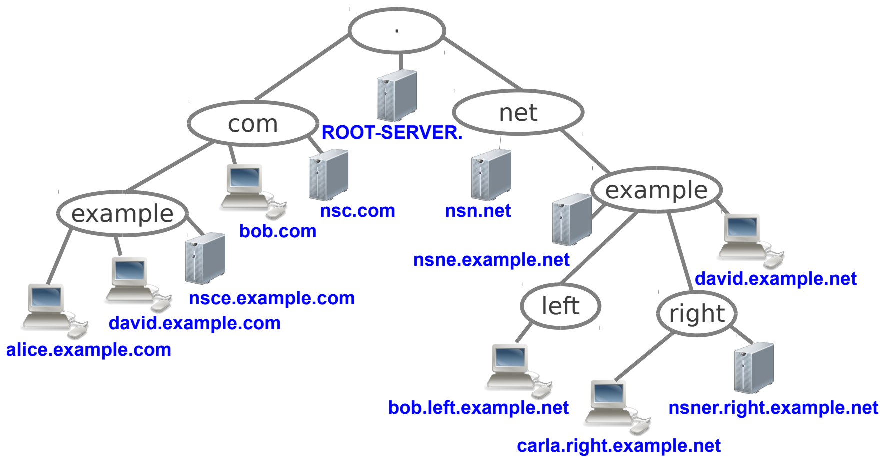


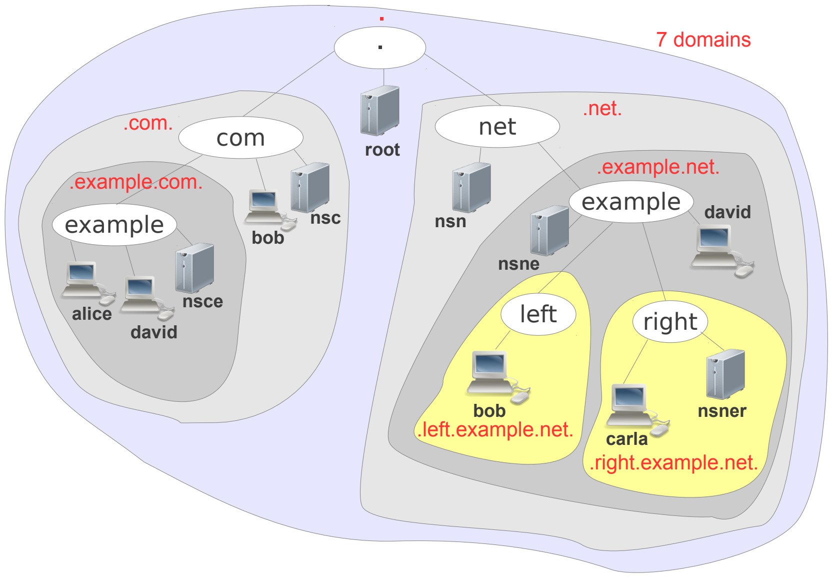
Figure 1.4: A Sample DNS Tree.

As it can be observed in the Figure 1.5a, the DNS tree of Figure 1.4 has 7 domains: dot, .com, example.com, .net, example.net, left.example.net and right.example.net. Each domain has one or more leaves (network devices). For instance, the domain .com has two leaves: bob.com and nsc.com.

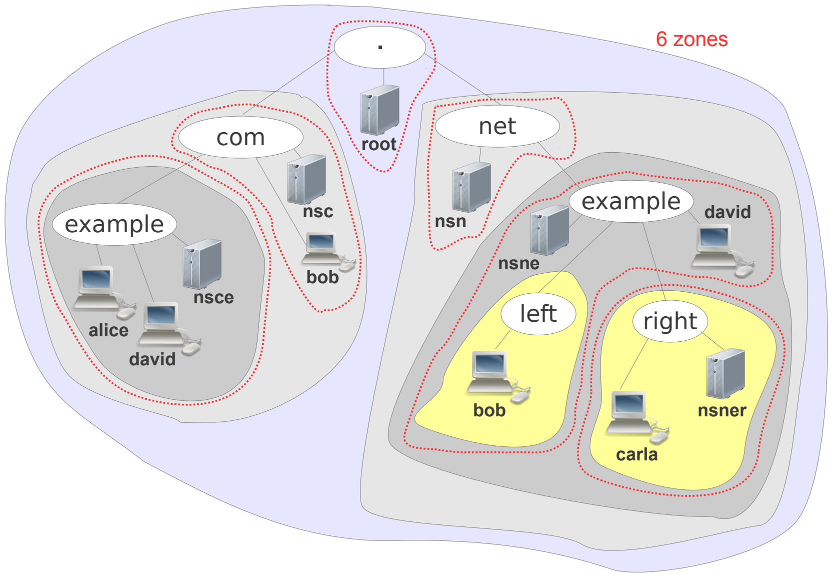
1.2.2 Zones

For delegation, the DNS tree is administratively divided into **zones**. A zone is a delegation and administration point. Simplifying a zone is a **configuration file** containing a set of translations that is managed by a master or primary name server. If the zone is big or if some degree of redundancy is desired, we can also use secondary servers in a master/slave configuration. Zones do not overlap, which means that the translation of a leaf of the DNS tree is only present at the corresponding primary server or secondary servers of the zone. For example, the DNS tree of Figure 1.4, which has 7 domains can be administratively implemented with a different number of zones and servers.

A server can serve multiple zones but, for the sake of simplicity, by now, we will consider that each zone is implemented with just one server and that each server just serves one zone. Figure 1.5b shows a possible implementation with 6 zones and 6 primary servers.



(a) Domains.



(b) Zones.

Figure 1.5: A Sample DNS Tree: Domains and Zones.

As you can observe in Figure 1.5, the server **root** is in charge of the zone dot ("."). That is, the server **root** has the translations of all the names except those that it has delegated. In our sample DNS tree, the **root** server delegates .com and .net. The domain .com is managed by **nsc** and the domain .net is managed by **nsn**. Regarding the server **nsc**, this server manages all the names ending with .com except those ending with example.com. The domain example.com has been delegated to a server called **nsce**. Thus, the translation of bob.com is stored in **nsc** but the translation of alice.example.com is stored in **nsce**. On the other side, **nsn** delegates the administration of the names ending with example.net to the server **nsne**. In turn, **nsne** delegates the names ending with right.example.net to the server **nsner** but the rest of the names ending with example.net are resolved by **nsne**. In our example, names ending with example.net and left.example.net are defined in a single zone served by **nsne**. We must remark that we could also use two different zones: one for example.net and another for left.example.net both served by **nsce**. In any case, **nsce** has the translations for david.example.net and bob.left.example.net but not the translation for carla.right.example.net which is stored in the server **nsner**.

On the other hand, the DNS hierarchy is orthogonal with respect to the network topology or network addressing. In our example (see Figure 1.6), we have just a single IP network: 10.0.0.0/24. Obviously, in the "real" Internet we have many different IP networks connected with routers. Nevertheless, these networks are not necessarily related to the DNS tree.

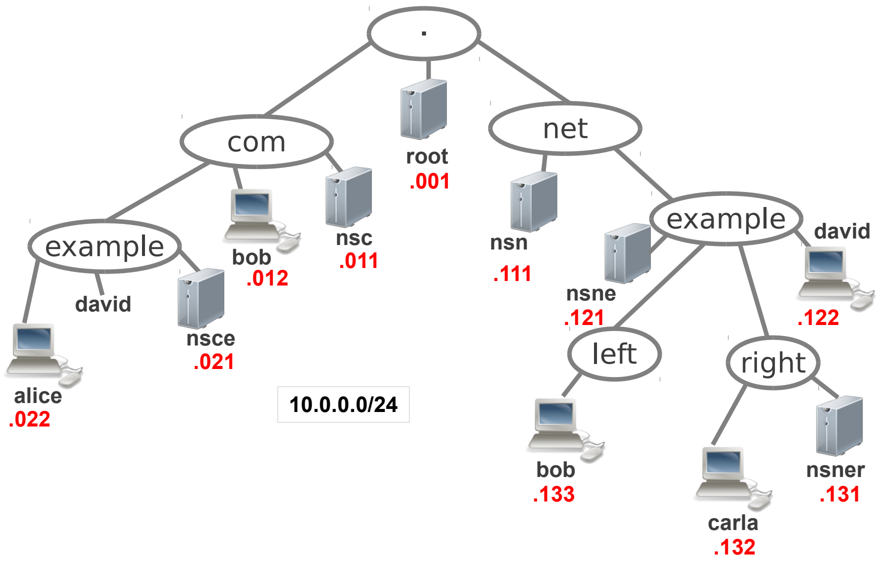


Figure 1.6: A Sample DNS Tree IPs.

Notice that david.example.com does not have an IP address assigned. This is because this name is in fact another name for david.example.net which is assigned to the address 10.0.0.122.

1.3 Implementation

1.3.1 Resource Records

The DNS database is implemented with the so called Resource Records (RRs). RRs are text lines that define the configuration of the DNS tree nodes. The general format of RR is the following:

```
Owner   [TTL]    Class    Type    RDATA
```

You need to use parenthesis if the RR does not fit into a single line. The meaning of each field is the following:

- **Owner**: RR owner, i.e. a name.
- **TTL**: the time that a RR may be cached by any resolver (optional).
- **Class**: Resource records belong to a class. Typically, the class is IN (Internet).
- **Type**: There are different types of records, e.g. Type = A is an IP address.
- **RDATA**: Record Information. For example, the A record contains an IP address.

1.3.2 Resource Record “A”

One of the most used resource records is the A record. This RR contains the IP Address associated with a name. For example, the A resource record for the name alice.example.com with a TTL for caching of 30 seconds is the following:

```
alice.example.com. 30 IN A 10.0.0.22
```

1.3.3 Resource Record “SOA”

Another important RR is the Source of Authority (SOA) record. The SOA is always the first RR of a zone and it contains administrative information. Each zone must have a different SOA. The RDATA of this record is the following:

- **Origin**: name of the zone’s primary server.
- **Person**: e-mail of the zone’s administrator.
- **Serial**: integer (YYYY/MM/DD/XX) that must be increased after any modification of the zone data.
- **Refresh**: time between zone transfer requests by secondary servers (usually days).
- **Retry**: time between requests whenever a zone transfer request fails (usually hours).
- **Expire**: time a secondary server keeps the data if the connection to the master fails (usually months).
- **Negative cache**: the time that an inexistent translation may be cached.

Following our example, the SOA for the zone example.net can be something like the following:

```
example.net.  IN  SOA  nsne.example.net.  admin-mail.nsne.example.net. (
2006031201 ; serial
28800 ; refresh
14400 ; retry
3600000 ; expire
0 ; negative cache ttl
)
```

The previous SOA says that the primary server of the zone example.net is nsne.example.net and that the e-mail of the zone administrator is admin-mail@nsne.example.net. The serial number of the SoA is 2006031201. If the primary server changes its configuration (adds, removes or changes any RR) the serial number has to be incremented. Then, the secondary servers will contact the primary server every 8 hours (28800 seconds) to check for changes. If a zone transfer fails, the secondary server will try again in 4 hours (14400 seconds). In case the connection with the primary server is not possible, the secondary will keep the data for ≈42 days (3600000 seconds). The final parameters says that if a name has not a translation on the server, the server will return a special response saying that this name has not a translation and that this event is not cached (negative cache = 0).

Remark. If we use only one zone for all the names ending with example.net, then we should define names like bob.left.example.net in the zone example.net. Another possibility is to define a different zone (also served by nsne.example.net) for the domain names ending with left.example.net. In this last case, we have to define another SOA.

1.3.4 BIND

The most popular implementation of DNS is the Berkeley Internet Name Domain (BIND), which is maintained by http://www.isc.org. Some of the most relevant features of BIND are the following:

- Names to IP addresses translation.
- Inverse Translation.
- Access Control Lists (ACL).
- Secondary servers.
- Secure zones transfers between primary and secondary servers (and ports).
- Service Location (SRV records).
- Parameterized replies depending on the origin of the request (views).
- Custom logs.

Next, we use `bind` to implement the configuration of our example DNS tree (see Figure 1.5), which emulates a “mini” Internet. In Linux, you can start, stop or restart `bind` using the command:

```
$ /etc/init.d/bind9 stop/start/restart
```

Each time you change the configuration of `bind` you have to restart it so that changes take effect.

On the other hand, it is worth to know that BIND writes its logs by default in a Debian distribution in the file /var/log/daemon.log.

1.3.5 Configuration of root

Let us start explaining the configuration of server **root** (see Figure 1.5). This server contains the configuration of the dot (".") zone. In general, the configuration files of `bind` are in the directory /etc/bind. The first configuration file that `bind` consults is /etc/bind/named.conf. In **root**, the contents of this file are the following:

```
options {
directory "/var/cache/bind";
};
zone "." {
type master;
file "/etc/bind/db.root";
};
zone "localhost" {
type master;
file "/etc/bind/db.local";
};
zone "0.0.10.in-addr.arpa" {
type master;
file "/etc/bind/db.10.0.0";
};
```

Let us observe now the configuration of the zone ".". In the file /etc/bind/named.conf you can observe that the configuration of this zone is specified in the file /etc/bind/db.root. For our example, we have used the following configuration in /etc/bind/db.root:

```
$TTL    60000 ; 16h40m default Time to Live of the DNS records
.   IN    SOA    ROOT-SERVER.   admin-mail.ROOT-SERVER.(
2006031201 ; serial
28800 ; refresh
14400 ; retry
3600000 ; expire
0 ; negative cache ttl
)

.              IN   NS   ROOT-SERVER.
ROOT-SERVER.   IN   A    10.0.0.1

com.           IN NS     nsc.com.
nsc.com.       IN A      10.0.0.11

net.           IN   NS    nsn.net.
nsn.net.       IN   A     10.0.0.111
```

As you can observe there are resource records A and SoA (already mentioned). Notice that in this zone all the names end with a dot ".". This means that all the names are FQDN. Comments can be included in the configuration files after a semicolon (";") or double slash ("//"). Finally, you can observe that there is a new resource record: the NS (Name Server). This record and the previous configuration is explained next.

1.3.6 Resource Record "NS"

NS records are used for delegation. Each NS links a domain name with the name of an authoritative name server for that domain. All the zones start with a SOA and a NS record.

On the other hand, the configuration file of the zone will also have as many NS records as domains being delegated.

In our previous configuration file, the first NS record says that the root zone "." is served by a server called ROOT-SERVER. Then it follows an A record showing that the IP address associated with ROOT-SERVER. is 10.0.0.1. Then, we can observe that we have one NS record saying that the domain .com. is being delegated to a server called nsc.com. and there is another NS record saying that the zone .net. is being delegated to a server called nsn.net.

1.3.7 Glue Records

Notice that there are two A records accompanying the two NS records of .com and .net. This is because the NS record does not contain any IP address and in principle, the name server of .com, which is nsc.com, should contain all the translations of names ending with .com. This includes its own name, nsc.com. So if we do not include an A record in the **root** we are in a loop:

*"The **root** server is delegating the subdomain .com to a server called nsc.com and in principle the translation of this name has to be at nsc.com."*

Thus, only with the NS record of .com there is no way to obtain the IP address of its server nsc.com. For this reason, we need to include the A record (called glue record) of the delegated nameserver in the parent zone. Notice however, that the glue record is only necessary when the server of the delegated zone has a name within the delegated domain, but the glue record is not necessary if the name server to which a subdomain is being delegated does not have a name in that subdomain. For example:

```
.               IN   NS   ROOT–SERVER.
ROOT–SERVER.    IN   A     10.0.0.1

com.            IN  NS     nsc.com.
nsc.com.        IN  A      10.0.0.11  ; needs glue record

net.            IN   NS     nsn.com.  ; does not need glue record
```

In the previous configuration, notice that we do not need a glue record for .net.

1.3.8 Configuration of nsc.com

Let us observe now the configuration of the server nsc.com. Recall that the main configuration file of `bind9` is /etc/bind/named.conf. We have the following configuration in this file:

```
options {
directory "/var/cache/bind";
min−roots 1;
};
zone "." {
type hint;
file "/etc/bind/db.root";
};
....
// add entries for other zones below here
zone "com" {
type master;
file "/etc/bind/db.com";
};
```

The file starts with some options and then follows "hint information" about the zone ".". This hint information is described next.

1.3.9 Root Servers

Any DNS server must have information about how to reach the root servers of the DNS tree. This information is sometimes called "root hints". Notice that in our configuration, root hints are specified in the file /etc/bind/db.root. As we will explain in Section 1.7, in the Internet there are 13 root servers but in our example (see Figure 1.5), we only have one root server. We have configured the file /etc/bind/db.root with the following contents:

```
.                    IN   NS     ROOT–SERVER.
ROOT–SERVER.         IN   A      10.0.0.1
```

Root hints are necessary because servers of the DNS tree can “translate down“ but they cannot not ”translate up”. In other words, a server knows how to do a translation for any name under one of its subdomains (either directly or via another nameserver), but the nameserver must contact a root server to be able to reach parts of the DNS tree that are not directly below it.

1.3.10 Configuration of a Zone

Now let us observe the configuration of the zone “.com” which is specified in the file /etc/bind/db.com at nsc.com. In our example, we have used the following configuration in this file:

```
$TTL   60000 ; 16h40m default Time to Live of the DNS records
com.  IN  SOA  nsc.com.  admin−mail.nsc.com. (
2006031201 ; serial
28800 ; refresh
14400 ; retry
3600000 ; expire
0 ; negative cache ttl
)
com.                IN    NS     nsc.com. ; ns of .com
nsc.com.            IN    A      10.0.0.11 ; leaf of .com
bob.com.    30      IN    A      10.0.0.12 ; leaf of .com
example.com.        IN    NS     nsce.example.com. ; delegation of example.com
nsce.example.com.   IN    A      10.0.0.21 ; glue record of example.com
```

The file contains the TTL parameter (set to 16h40m), the SOA record and the NS record of .com. Then, there are A records for nsc.com and bob.com, which are leaves of this zone. Observe that the A record for bob.com has specified a particular TTL of 30 seconds (this record does not use the default value of the TTL parameter). This means that the A record of bob.com is going to be cached only during 30 seconds by any resolver. Then, the subdomain example.com has been delegated to another nameserver called nsce.example.com. We need the glue record, which points to the address 10.0.0.21. Finally, notice that all the names are FQDN (end with a dot).

1.3.11 Configuration of nsce.com

Now, let us discuss the configuration of nsce.example.com. The named.conf of this servers is:

```
options {
directory "/var/cache/bind";
min−roots 1;
};
zone "." {
type hint;
file "/etc/bind/db.root";
};

zone "localhost" {
type master;
file "/etc/bind/db.local";
};
...


// add entries for other zones below here
zone "example.com" {
type master;
file "/etc/bind/db.com.example";
};
```

Let us see the configuration of the zone “.example.com” in the file /etc/bind/db.com.example:

```
; /etc/bind/db.com.example
$ORIGIN example.com.
$TTL     60000
@        IN    SOA    nsce   admin-mail.nsce (
2006031201 ; serial
28 ; refresh
14 ; retry
3600000 ; expire
20 ; 20 secs of negative cache ttl
)
@                    IN   NS      nsce       ; unqualified name
nsce                 IN   A       10.0.0.21
david                IN   CNAME   david.example.net.
@                    IN   MX    10 mailserver1
@                    IN   MX    20 mailserver2.example.com.
alice                IN   A       10.0.0.22
mailserver1          IN   A       10.0.0.25
mailserver2          IN   A       10.0.0.26
; alice.example.com  IN   A       10.0.0.22
```

The configuration of this zone has some details that are worth to mention. First, as you can see there is a parameter called $ORIGIN. This parameter is used to append its content (a domain name) to unqualified names (names that do not end with a dot). In the previous configuration, example.com is appended to any name without trailing dot. The default value for the $ORIGIN is the zone name specified in /etc/named.conf. Thus, in this case, the directive $ORIGIN is unnecessary. The sign @ is used as a substitute of the value of $ORIGIN. In this context, notice that if we uncomment (remove the semicolon) of the ultimate line we are defining an A record for the name alice.example.com.example.com. Using unqualified names is useful to easily rename zones. For example, with a single change in the named.conf file, we can change all the names of a subdomain to a different one.

Finally, notice that in the previous configuration file there are two new RRs: CNAME and MX.

1.3.12 Resource Record “CNAME“

The record CNAME is used to create an alias to a canonical name (CNAME). This RR associates a name with another name of the DNS tree (see Figure 1.7). This is useful for creating a configuration with several names translating to the same IP address. Notice that for this purpose, another possible configuration is to use several A records. Example:

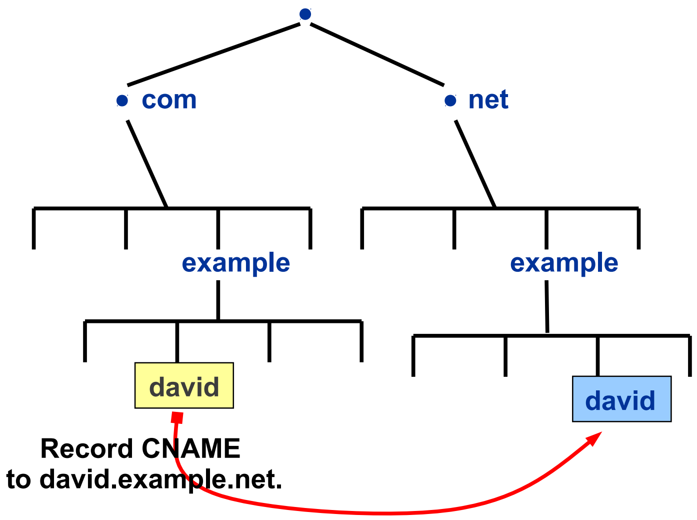


Figure 1.7: CNAME Register.

1.3.13 Resource Record ”MX“

The MX (Mail eXchanger) records designate the mail servers for a given domain. These RRs allow finding the location of mail exchange servers. Example:

```
@                    IN   MX   10  mailserver1
@                    IN   MX   20  mailserver2.example.com.
```

In this example, any email to an address of the form **XXX@example.com** is going to be served either by **mailserver1** or by **mailserver2**. The e-mail servers will be contacted by priority order (lower number first).

Note. For other services different from the e-mail, there are generic RRs called SRV (service record, see Section 1.9.3).

1.3.14 Services and DNS

An interesting application of the DNS is that it can be used to build a load sharing mechanism for services. The idea is very simple: use two A registers for the same name. Then, the name server provides alternate addresses to the clients using a round robin mechanism. This is useful to share the load between two servers. Example:

```
www                  IN   A      10.0.0.27
www                  IN   A      10.0.0.28
```

Another interesting application of the DNS is to do translations depending on the source address. In this way, we can send traffic to a server or to another for providing a specific treatment depending on the source.

1.4 DNS Queries/Responses

1.4.1 Client Configuration

The first that a Linux application does when trying to find a translation for a name is to look at the local translations (remember that these are defined in /etc/hosts). If there is not any translation in this file, the client has to use the **DNS service**. To do so, we need to know at least one name server. In Linux, the file /etc/resolv.conf is where the addresses of the nameservers are stored. The contents of this file are something like the following:

```
nameserver 147.83.2.3
nameserver 147.83.2.10
search upc.edu
```

The format of this file is as follows:

- `nameserver @IP`. Points out to a nameserver to which the client will ask for DNS translations. In the resolv.conf file you can set up to 3 nameservers that will be contacted in order. Obviously, the nameserver to which you ask, has to be configured to accept the client queries. For this purpose, filters based on IP addresses can be defined for example with `bind` to control which clients can access and which not. On the other hand, the name server is typically provided by your ISP or your organization[1].
- `search upc.edu`. Users can use unqualified names if the search primitive is used. In the previous example, the search is upc.edu. You can use a trailing dot in the name if you do not want to use the search in your application.

In Linux, we have a couple of useful commands to perform DNS queries from clients: `host` and `dig`. Examples:

```
alice$ host alice
alice$ host alice.example.com
alice$ dig  alice
alice$ dig  alice.example.com
```

Note. The `dig` command does not use unqualified names, i.e. it never uses the “search”.

[1]There are also “open” name servers like 8.8.8.8 and 8.8.4.4 (from Google) that anyone can use.

1.4.2 Queries

Each query specifies the domain name, class and type of the queried RR. For example, we can ask for `www.upc.edu  IN  A`.

There are two types of queries: **iterative** and **recursive**.

- In an iterative query, the queried name server returns the address of the next name server that you must consult to obtain the response.
- In a recursive query, the name server tries to find the final response, making several queries to different name servers if necessary. If we want to do a recursive query, we have to activate the flag recursion flag on the request. In addition, the server must support recursion.

Typically, clients perform recursive queries and name servers perform iterative queries to other name servers. A name server can also act as recursive for a given set of nodes (IP networks) and as non-recursive for the rest.

1.4.3 Types of Responses

Name servers typically use a cache to improve the DNS performance: a name server can use a response for a previous query to answer subsequent queries. The time that a response is cached is configured by the original source administrator with the general TTL parameter or with the per RR TTL, which overrides the general one. When we use cache, we have two possible types of responses: **authoritative** and **non authoritative**. An authoritative name server provides authoritative responses (not cached) because it is the original source. A non-authoritative name server provides cached responses (with a valid TTL). In general, cached RR are used by servers but final clients do not use cache.

1.5 DNS Protocol

DNS servers use by default the UDP port 53 and DNS messages (queries and responses) might have five parts: header, questions, answers, authority and additional info (see Figure 1.8).

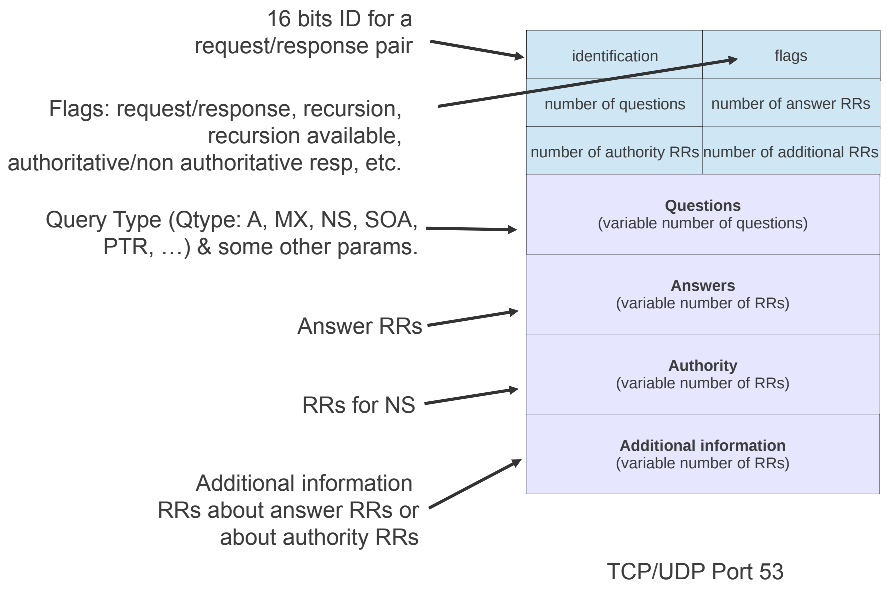

16 bits ID for a request/response pair
Flags: request/response, recursion, recursion available, authoritative/non authoritative resp, etc.
Query Type (Qtype: A, MX, NS, SOA, PTR, …) & some other params.
Answer RRs
RRs for NS
Additional information RRs about answer RRs or about authority RRs
identification
flags
number of questions
number of answer RRs
number of authority RRs
number of additional RRs
Questions
(variable number of questions)
Answers
(variable number of RRs)
Authority
(variable number of RRs)
Additional information
(variable number of RRs)
TCP/UDP Port 53

Figure 1.8: DNS protocol.

- **Message header**. In the header we can find if the message is a query or a reply, if recursion is desired or not, if the response is authoritative or not, etc.
- **Questions**. Questions are always tuples with Name, Type and Class. The Name is a fully-qualified domain name. Names cannot be wildcarded, but Types and Classes can be. In addition, special Types exist to wildcard

mail records and to trigger zone transfers. The question is the only section included in a query message. The remaining sections are used for replies.

- **Answers**. The answers part contains the registers that match the Name, Type and Class tuple asked in the questions part. If any of the matching records are CNAME pointers leading to other records, the target records should also be included in the answer. There may be multiple answers, since there may be multiple RRs with the same labels.
- **Authority**. The authority part typically contains NS records pointing to name servers closer to the target name in the naming hierarchy. This field is optional, but resolvers are encouraged to cache NS records in this part to make further resolutions more efficient. This is because there can be other further questions that need the same nameserver and thus, it is useful to have the NS cached.
- **Additional information**. This part contains additional records that the name server believes may be useful to the client. The most common use for this field is to supply A (address) records for the name servers listed in the Authority section. However, more clever name servers are feasible. For example, let us consider that we request the MX record of example.com. and the answer points to mail.example.com. The name server can infer that the client's next request will be an A query for mail.example.com. Let us consider that mail.example.com contains a CNAME record pointing to saturn.example.com. Then, the name server can avoid all this extra traffic by just including the CNAME and A records as additional RRs in the original reply.

1.6 Practical Name Resolutions

1.6.1 Querying an Authoritative Server

Let us start discussing the query of a client to an authoritative server. In our example, let us consider that Alice has 10.0.0.21 (which is **nsce**) as her resolver in /etc/resolv.conf and that she wants to know its own IP address (see Figure 1.9):

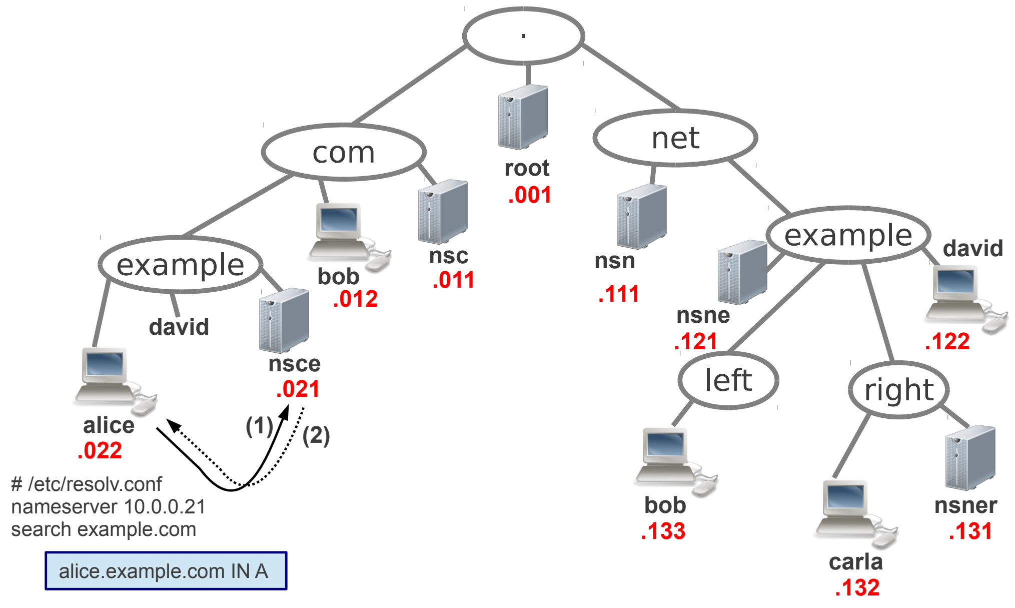


Figure 1.9: Querying an Authoritative Server.

(1) **alice** sends a recursive query for the A register of alice.example.com to **nsce**.
(2) **nsce** is an authoritative server for the name alice.example.com, therefore, the server will not need to contact any other server and the response will be authoritative.

To generate a request and receive the corresponding response we can use the following command:

```
alice$ dig alice.example.com
; <<>> DiG 9.6-ESV-R4 <<>> alice.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49052
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;; QUESTION SECTION:
;alice.example.com.             IN      A
;; ANSWER SECTION:
alice.example.com.      60000   IN      A       10.0.0.22
;; AUTHORITY SECTION:
example.com.            60000   IN      NS      nsce.example.com.
;; ADDITIONAL SECTION:
nsce.example.com.       60000   IN      A       10.0.0.21
;; Query time: 289 msec
;; SERVER: 10.0.0.21#53(10.0.0.21)
;; WHEN: Tue Mar 26 11:29:34 2013
;; MSG SIZE  rcvd: 86
```

The output of the previous command shows all the sections of the DNS response. Also the flags tell us that the response is authoritative (aa), that recursion was desired on the request (rd) and it was available on the server (ra). Using a protocol analyzer (like `wireshark`) you can observe which are the DNS protocol messages that this command generates. The request is as follows:

```
Internet Protocol Version 4, Src: 10.0.0.22 (10.0.0.22), Dst: 10.0.0.21 (10.0.0.21)
User Datagram Protocol, Src Port: nfs (2049), Dst Port: domain (53)
Domain Name System (query)
[Response In: 4]
Transaction ID: 0xbf9c
Flags: 0x0100 (Standard query)
0... .... .... .... = Response: Message is a query
.000 0... .... .... = Opcode: Standard query (0)
.... ..0. .... .... = Truncated: Message is not truncated
.... ...1 .... .... = Recursion desired: Do query recursively
.... .... .0.. .... = Z: reserved (0)
.... .... ...0 .... = Non-authenticated data: Unacceptable
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
alice.example.com: type A, class IN
Name: alice.example.com
Type: A (Host address)
Class: IN (0x0001)
```

As you can see the client is asking for a recursive request. The corresponding response from the server **nsce** is as follows:

```
Domain Name System (response)
[Request In: 3]
[Time: 0.000774000 seconds]
Transaction ID: 0xbf9c
Flags: 0x8580 (Standard query response, No error)
1... .... .... .... = Response: Message is a response
.000 0... .... .... = Opcode: Standard query (0)
.... .1.. .... .... = Authoritative: Server is an authority for domain
.... ..0. .... .... = Truncated: Message is not truncated
.... ...1 .... .... = Recursion desired: Do query recursively
.... .... 1... .... = Recursion available: Server can do recursive queries
.... .... .0.. .... = Z: reserved (0)
.... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated
.... .... ...0 .... = Non-authenticated data: Unacceptable
.... .... .... 0000 = Reply code: No error (0)
```

```
Questions: 1
Answer RRs: 1
Authority RRs: 1
Additional RRs: 1
Queries
alice.example.com: type A, class IN
Name: alice.example.com
Type: A (Host address)
Class: IN (0x0001)
```

Observing the flags of the response you can see that the response is authoritative (aa), that recursion was desired (rd) and it was also available (ra) on the server. In the response you can also see that there is one question (the A record for alice.example.com). For that question there is an answer (the corresponding A record) but also there are records in the authority (NS record of nsce.example.com) and additional parts (A record of nsce.example.com). The response continues with the following fields:

```
Answers
alice.example.com: type A, class IN, addr 10.0.0.22
Name: alice.example.com
Type: A (Host address)
Class: IN (0x0001)
Time to live: 16 hours, 40 minutes
Data length: 4
Addr: 10.0.0.22 (10.0.0.22)
Authoritative nameservers
example.com: type NS, class IN, ns nsce.example.com
Name: example.com
Type: NS (Authoritative name server)
Class: IN (0x0001)
Time to live: 16 hours, 40 minutes
Data length: 7
Name Server: nsce.example.com
Additional records
nsce.example.com: type A, class IN, addr 10.0.0.21
Name: nsce.example.com
Type: A (Host address)
Class: IN (0x0001)
Time to live: 16 hours, 40 minutes
Data length: 4
Addr: 10.0.0.21 (10.0.0.21)
```

1.6.2 Non-existent Name

Now, let us try a query for a non-existent target:

```
alice$ dig alan.example.com
; <<>> DiG 9.6-ESV-R4 <<>> alan.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 39136
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
;; QUESTION SECTION:
;alan.example.com.              IN      A
;; AUTHORITY SECTION:
example.com. 20 IN SOA nsce.example.com. ...
```

As you can see, the server returns a response with status NXDOMAIN, which means that a translation for the name was not found. The server also returns the SOA because this RR contains the negative TTL, which tells us how long we must store in cache this information. In this case, 20 seconds.

1.6.3 Non authoritative server

The **nsce** server is not an authoritative server for the name bob.com. The authoritative server for this name is **nsc**. In this case, let us consider what happens when **alice** asks **nsce** for the A record of bob.com.

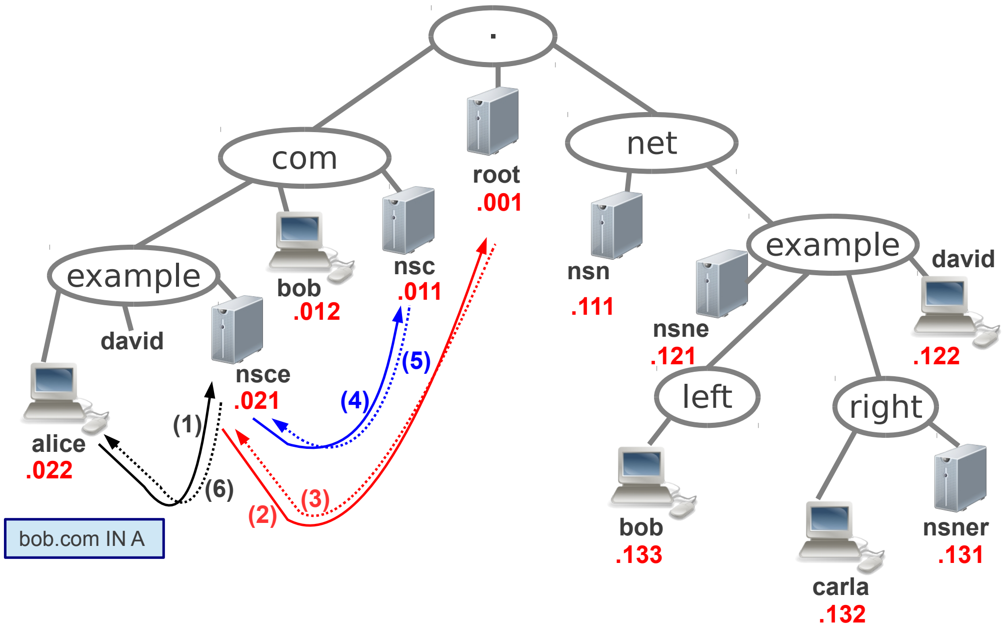


Figure 1.10: Query an Non Authoritative Server.

To test this situation, you can use `dig` to generate the query and capture with `wireshark`. Then, using the captured traffic you can create a message flow graph as the following by clicking over a packet of the flow and then using the `wireshark` option *Statistics->Flow Graph*.

```
alice.example.com   nsce.example.com    ROOT-SERVER      nsc.com

| 10.0.0.22                             | 10.0.0.1
|                   | 10.0.0.21         |                   | 10.0.0.11
|         Query A bob.com               |                   |
|(2049)   ---------->  (53)             |                   |
|                   |         Query A bob.com               |
|                   |(10462)  ----------->  (53)            |
|                   |         Query NS <Root>               |
|                   |(42735)  ----------->  (53)            |
|                   |         Resp NS,A nsc.com             |
|                   |(10462)  <------------  (53)           |
|                   |         Resp NS,A ROOT-SERVER         |
|                   |(42735)  <------------  (53)           |
|                   |         Query A  bob.com              |
|                   |(6107)   ------------------------------>  (53)
|                   |         Resp  A 10.0.0.12             |
|                   |(6107)   <------------------------------  (53)
|         Resp  A 10.0.0.12             |                   |
|(2049)   <---------  (53)              |                   |
```

As shown in the flow graph, **nsce** finds the authoritative server iteratively. Since bob.com is not down in the DNS hierarchy of **nsce**, **nsce** will start the search from the root of the DNS tree. This is the reason why a name server must know always at least one root server. In our example, when **alice** asks her name server the translation for bob.com (see Figure 1.10):

(1) **alice** sends a recursive query for the A register of bob.com to **nsce**.

(2) **nsce** sends an iterative query for the A record of bob.com to **root**. Actually, the **nsce** also asks for the NS record of the **root**.
(3) **root** provides its NS/A and the NS/A records of **nsc**, which is who serves .com.
(4) Then, **nsce** sends an iterative query to **nsc** for A record for bob.com,
(5) Then, **nsc** sends the A register of bob.com to **nsce**.
(6) Finally, **nsce** sends the A register of bob.com to **alice**.

As you can observe the process to obtain a DNS translation uses the network intensively and this process happens extremely often in the Internet. To reduce the cost of the resolution process "caching" is used.

1.6.4 DNS Caching

Name servers use caching to enhance the performance of the DNS system by storing the RRs obtained during the resolution processes. These RRs are kept in cache until their TTL expires. A non authoritative server can use its cached RRs with valid TTL to answer to its clients' queries without further contacting any other name servers.

In our example, if **alice** or another user asks **nsce** for the A record bob.com and **nsce** still has this record in its cache (recall that we configured this record with a TTL of 30 seconds), then, **nsce** can directly respond the query with its cached record (see Figure 1.11).

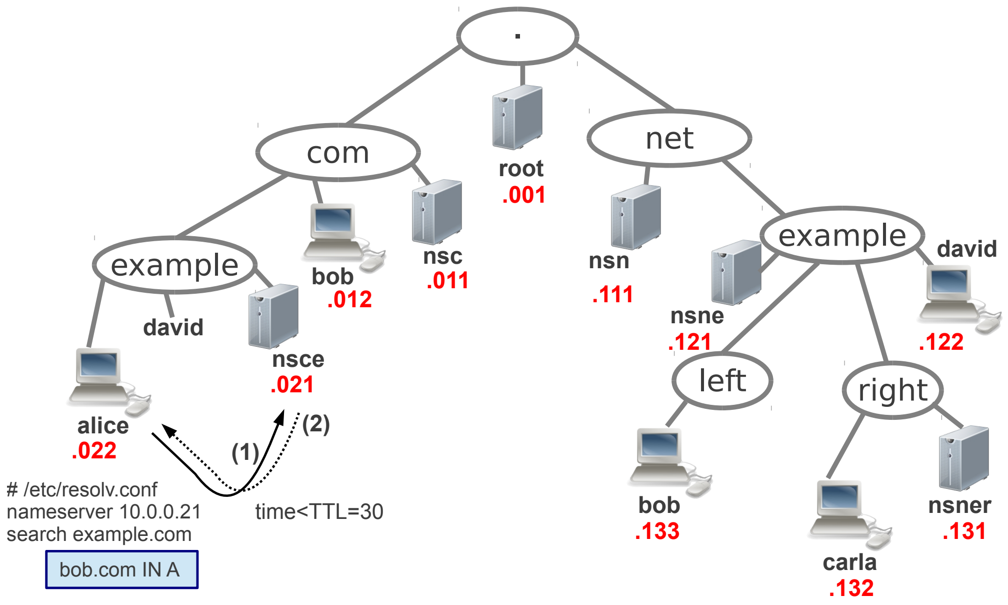


Figure 1.11: Query an Non Authoritative Server with Information in Cache.

```
alice$ dig bob.com
; <<>> DiG 9.6-ESV-R4 <<>> bob.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52900
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;; QUESTION SECTION:
;bob.com.                        IN      A
;; ANSWER SECTION:
bob.com.                17       IN      A       10.0.0.12
;; AUTHORITY SECTION:
com.                    59679    IN      NS      nsc.com.
```

Now, let us consider that **alice** asks **nsce** for the A record of bob.com, but **nsce** observes that the TTL of its record has expired:

```
alice$ dig bob.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4345
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;; QUESTION SECTION:
;bob.com.                       IN      A
;; ANSWER SECTION:
bob.com.                30      IN      A       10.0.0.12
;; AUTHORITY SECTION:
com.                    59447   IN      NS      nsc.com.
```

In this case, **nsce** can still use the NS record and the A (glue) record of nsc.com since these records are not yet expired. Using these records, **nsce** can avoid to contact **root** but it can directly contact **nsc** which from the NS record we know that is the server for the .com domain (see Figure 1.12). This can be checked in practice observing the output of the previous `dig` command. You can observe that **nsce** has reused the NS record of **nsc** because the TTL of the NS record has decreased, but it got a brand new A record for bob.com because you can see a TTL of 30 seconds for this record.

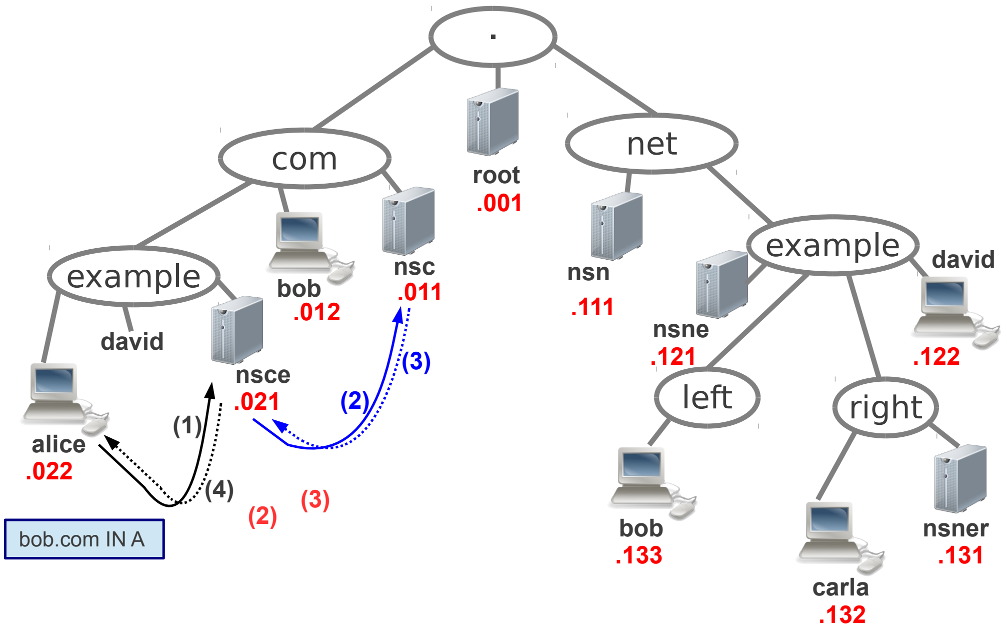


Figure 1.12: Query an Non Authoritative Server with Partial Information in Cache.

1.6.5 Non Recursive Questions

Remember that in `bind`, the list of ROOT-SERVERS is configured as follows:

```
; /etc/bind/named.conf
options {
directory "/var/cache/bind";
min-roots 1;
};
zone "." {
type hint;
file "/etc/bind/db.root";
};
...
```

min-roots is the minimum number of root servers. Default is 2, but in our example we only have configure one root server. The file db.root contains the NS and A records of the root server:

```
; /etc/bind/db.root
.                    IN   NS     ROOT–SERVER.
ROOT–SERVER.         IN   A      10.0.0.1
```

Let us try an iterative (non recursive) query with `dig` from **alice** to **nsce** about bob.com (for convenience, we will avoid displaying question section in the output of `dig`):

```
alice$  dig +noquestion +norecurse bob.com
; <<>> DiG 9.6-ESV-R4 <<>> +noquestion +norecurse bob.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30988
;; flags: qr ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
;; AUTHORITY SECTION:
.               0       IN      NS      ROOT-SERVER.
```

As you can observe, **nsce** returns the NS record of the ROOT-SERVER (**root**), which is the next server you have to contact to resolve bob.com. Now, if **alice** asks **root** for bob.com:

```
alice$  dig +noquestion +norecurse @10.0.0.1 bob.com
; <<>> DiG 9.6-ESV-R4 <<>> +noquestion +norecurse @10.0.0.1 bob.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23767
;; flags: qr ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
;; AUTHORITY SECTION:
com.                    60000   IN      NS      nsc.com.
;; ADDITIONAL SECTION:
nsc.com.                60000   IN      A       10.0.0.11
```

We obtain the NS and A records of **nsc**, which is the next (in fact the last) server to resolve bob.com.

1.6.6 Caching Servers

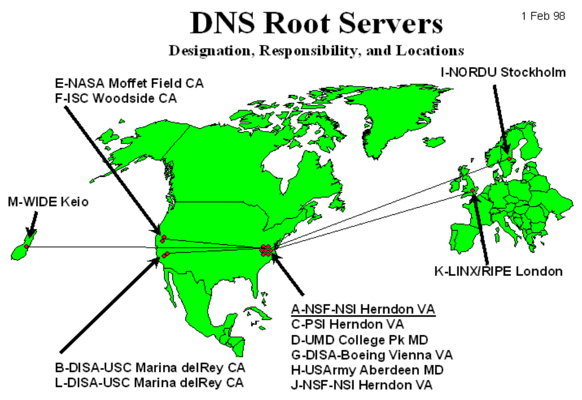
As a final remark, clients can be connected to a caching DNS server. This non authoritative servers do not contain any zone definitions but they are just used to answer to clients using caching, which enhances the resolution process. A caching server can be a forwarder or not a forwarder. The forwarder uses another DNS server (not a root server) to send its queries.

1.7 Real Root Servers

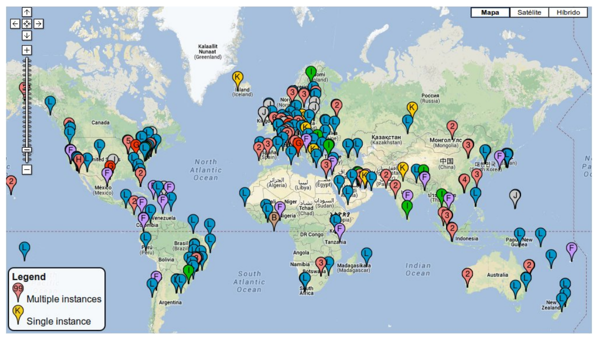
Obviously, in the Internet we do not have only one root server. Initially, there were 13 root servers. These servers were maintained by voluntary cooperation of organizations, had different names (including a letter from A to M) and were located in different continents (see Figure 1.13a).

Nowadays, we still have 13 names for root servers specified in the form letter.root-servers.net, where letter ranges from A to M. The reason of having exactly 13 root servers is the following. The RFC 791 specified that the minimum MTU is 576 Bytes. This value was chosen to allow a "reasonable sized" data block of at least 512 bytes, plus room for the standard IP header and options. We do not want fragmentation of DNS messages, thus, only 13 NS records and their corresponding A records fit into an UDP-based DNS message 512 bytes long.

However, this does not mean that we have only 13 physical servers. For reliability and performance each name server is implemented using redundant computer equipment to be able to provide the DNS service even if failure of hardware or software occurs. Additionally, some root servers operate in multiple geographical locations using a routing technique called **anycast**. Only 13 names and IP addresses are used but servers A, C, E, F, G, I, J, K, L and



(a) Old Root Servers.



(b) Actual Root-Servers.

Figure 1.13: Locations of Root Severs.

M servers now exist in multiple locations on different continents using anycast. As a result most of the physical root servers are now outside the United States, allowing for high performance worldwide (see Figure 1.13b[2]).

Anycast routing allows sending traffic to the nearest server of a certain service, in other words, two different network devices can send a packet to the same destination IP but the packet may end in a different destination. Anycast is implemented by routers.

Note. It must be remarked that there is no universal way to determine if an IP address is an unicast address or an anycast address just by looking at it.

1.8 Reverse Lookups

Simply stated, DNS lookups allow to translate a fully qualified domain name (FQDN) into an IP address, while reverse DNS lookups allow exactly the opposite, translating IP addresses into FQDNs. Defining the reverse translations is not compulsory. Some applications use the reverse translations as a measure of security but many applications do not use them. The resource record used in reverse lookups is called PTR (pointer). This RR receives this name because it is a pointer to a FQDN (see Figure 1.14).

A PTR record always begins with a special name of the form "**D.C.B.A.in-addr.arpa.**". The subdomain in-addr.arpa in the DNS tree has a special meaning for implementing reserve lookups. More specifically, the name D.C.B.A.in-addr.arpa. is reserved for the reverse lookup of the IPv4 address A.B.C.D. The typical way of implementing reverse lookups it to delegate the corresponding subdomains in-addr.arpa to the ISPs or entities that own each IP range and then, each ISPs or entity implements on its DNS servers the reverse translations of its corresponding IP ranges of addresses. For example, the subdomain 83.147.in-addr.arpa. is delegated to the UPC because the UPC owns the range 147.83. Then, in the DNS server of the UPC we have PTR records like the following one:

```
135.2.83.147.in-addr.arpa.  IN   PTR   www.upc.es
```

The previous PTR register is used to make the IP address 147.83.2.135 to point to the FQDN www.upc.es. You can use the dig command to ask for a reverse translation with the -x option:

```
$ dig -x 147.83.2.135
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43112
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
```

[2]The picture was extracted from www.root-servers.org.

```
;135.2.83.147.in-addr.arpa.      IN      PTR
;; ANSWER SECTION:
135.2.83.147.in-addr.arpa. 172755 IN     PTR     www.upc.es.
135.2.83.147.in-addr.arpa. 172755 IN     PTR     demoweb.upc.edu.
135.2.83.147.in-addr.arpa. 172755 IN     PTR     barcelonatech-upc.eu.
135.2.83.147.in-addr.arpa. 172755 IN     PTR     upc.es.
135.2.83.147.in-addr.arpa. 172755 IN     PTR     upc.cat.
135.2.83.147.in-addr.arpa. 172755 IN     PTR     upc.edu.
```

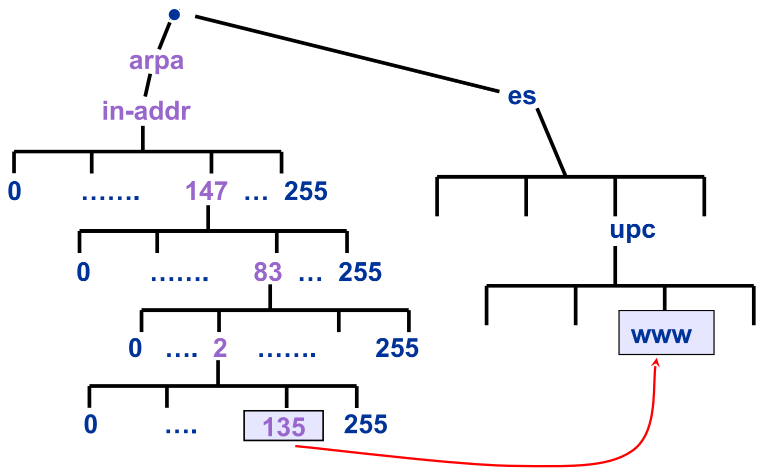


Figure 1.14: The PTR record.

As you can observe from the output of the previous command, we can have multiple PTR registers for a single IP address, that is to say, multiple FQDN for an IP address. In our “little Internet”, the reverse translations are implemented at **root**. Remember that the named.conf file of the **root** is the following:

```
......
zone "0.0.10.in-addr.arpa" {
type master;
file "/etc/bind/db.10.0.0";
};
```

The file db.10.0.0 contains the reverse lookup for the range of addresses 10.0.0.0/24:

```
$TTL     1
@     IN     SOA     ROOT-SERVER.   admin-mail.ROOT-SERVER. (
2006031201 ; serial
28 ; refresh
14 ; retry
3600000 ; expire
0 ; negative cache ttl
)
@       IN      NS      ROOT-SERVER.
1       IN      PTR     ROOT-SERVER.
11      IN      PTR     nsc.com.
12      IN      PTR     bob.com.
21      IN      PTR     nsce.example.com.
22      IN      PTR     alice.example.com.
111     IN      PTR     nsn.net.
121     IN      PTR     nsne.example.net.
122     IN      PTR     david.example.net.
131     IN      PTR     nsner.right.example.net.
132     IN      PTR     carla.right.example.net.
133     IN      PTR     bob.left.example.net.
```

1.9 Other RRs

1.9.1 Additional Info

The TXT record is a plaintext record containing optional information. Its use is free and arbitrary. It is shown in query result sets of the zone. Example:

```
TXT ``UPC DNS server''.
```

The HINFO record is optional and contains informative plain text about the server hardware and OS. Example:

```
HINFO ``Intel Pentium IV'' ``Debian Linux''.
```

The LOC record contains the geographical location of the server. This RR is informative and optional and it is used by graphical representation tools of server locations. Example:

```
LOC 39 34 58 N 2 38 2 E 100m 10000m 20m 100m.
```

1.9.2 Records for IPv6

There are two records for IPv6 addresses: AAAA and A6. The use of AAAA records is recommended, as A6 records have been made experimental by RFC 3363. The AAAA record is a parallel to the IPv4 A record. It specifies the entire address in a single record. For example:

```
example.host.com.  3600 IN AAAA 2001:db8:0:1::1
```

1.9.3 Resource Record "SRV"

The Service Resource Record (SRV) allows specifying the location of the servers of a certain service. These RRs are specified in RFC2052. Their format is the following:

```
_service._protocol.name TTL class SRV priority weight port destination
```

Where:

- Service: specifies the service name: sip, ldap, etc..
- Protocol: specifies the transport protocol to be used (e.g. TCP or UDP).
- Name: defines the domain name referenced by the SRV resource record.
- TTL & class has been previously defined.
- Priority: specifies the order in which clients will contact the servers.
- Weight: a relative weight for records with the same priority (load balancing).
- Port: port for the service at the server.
- Destination: complete domain name for the compatible hosts with the service.

Examples:

```
_ldap._tcp.upc.es. IN SRV 0 3 389  ldapserver1.example.com.
_ldap._tcp.upc.es. IN SRV 0 1 389  ldapserver2.example.com.
```

Note. The RFC for SRV records specifies that it may not be used by pre-existing protocols, which did not already specify the use of SRV records in their specifications. This is the reason why SRV records are not currently used by WEB browsers (HTTP protocol) or by the e-mail service. Examples of services using SRV records are SIP, LDAP, Kerberos, etc.

1.10 Administration Authorities

The Internet Corporation for Assigned Names and Numbers (ICANN[3]) coordinates several network parameters. In particular:

- It coordinates root and first level name servers.
- It coordinates the assignment of IP addresses.
- ICANN delegates zones to RIRs (Regional Internet Registries). We have six RIRs:
  - APNIC (Asia-Pacific Network Information Centre)
  - ARIN (American Registry for Internet Numbers)
  - LACNIC (Latin-American and Caribbean IP Address Registry)
  - RIPE NCC (Réseaux IP Européens)
  - AfriNIC (African Regional Internet Registry)

RIRs delegate the authority about domains to each country, so that the country or region can manage its domains. For example, for the .es domain has been delegated to “esNIC” (http://www.nic.es) and the .cat domain has been delegated to “puntcat” (http://www.domini.cat).

On the other hand, a RIR also assigns ranges of IP addresses to ISPs (or exceptionally to organizations) and implements the WHOIS service, which can be used to know to whom belongs an IP address. The WHOIS service uses the TCP port 43 on the server side. In Linux, you can use the `whois` command. Example:

```
$ whois 147.83.2.3
```

1.11 Extra*

1.11.1 Zone Transfer

Name servers that store information about a certain zone are known as authoritative servers. There can be multiple servers serving a zone. The purposes of using multiple authoritative servers are distributing the load and providing some failure tolerance. In this way, if an authoritative server crashes, the other servers can handle the requests. Authoritative servers must be synchronized. One authoritative server is the master (primary). The other authoritative servers are slaves (secondary). Secondary servers periodically fetch information from the master.

A procedure called **zone transfer** is necessary in those zones where exist several authoritative name servers (one master and one or more slaves). Every time changes are made in the master server, these changes must be mirrored on all secondary servers. There are two main types of zone transfers: full (AXFR) and incremental (IXFR). Secondary servers follow these steps when doing a zone transfer: The secondary waits during the refresh time of SOAs record. Asks the master for its SOA record. The master answers with its own SOA. The secondary server compares the received serial number with the one it already has. If it is greater, the secondary requests a zone transfer. Then, the master sends the zone database to the secondary server. If the master does not respond: The secondary will keep trying to contact the master every retry seconds. If there is not any response from the master after expiration seconds, the secondary server stops delivering the name service to the clients.

1.11.2 Dynamic DNS

DNS is usually static, which means that the translations have to be manually changed in the nameserver. On the other hand, we might want to link the public IP provided by our ISP to a name. We know that ISPs automatically assign an IP address to their clients (using DHCP) and these IPs might be changed after a network disconnection or reconnection. So, this is a problem for our DNS translation. Dynamic DNS (DDNS) is a method that allows you to update (add, delete or change) resource records in a given DNS Zone, without manual editing.

[3]http://www.icann.org

1.11.3 DNS Security Extensions

By default, DNS has no security. It is true that you can use IP-based authentication but it is easy to bypass this security. DNSSEC is a security extension set for the DNS protocol (new resource registers, flags, expansion of UDP-packet size and more). DNSSEC can use symmetric and asymmetric cryptography.

Transaction signatures or TSIG use symmectric cryptography (shared secrets) and a one-way hash function to authenticate DNS messages. TSIG is easy and lightweight. Each name server adds a TSIG record the data section of a dns server-to-server queries and message. The TSIG record signs the DNS message, proving that the message's sender had a cryptographic key shared with the receiver and that the message was not modified after it left the sender. TSIG uses a one-way hash function to provide authentication and data integrity. TSIG is used for zone transfers and for dynamic updates.

Finally, DNSSEC can also use public key cryptography and digital signatures. For such purpose were created the registers **DNSKEY**, **RRSIG** and **NSEC** Records. DNSEC provides us a mechanism to delegate the public key over the name server chain of DNS (chain of trust). This means that the key flows from parent Zone to child Zone and only Zone's parent can certify for its keys identity. For example, `.edu` signs upc.edu's key and the root signs .edu's key. In this case, you have **Delegation Signer** (DS) Record.

1.12 Practices

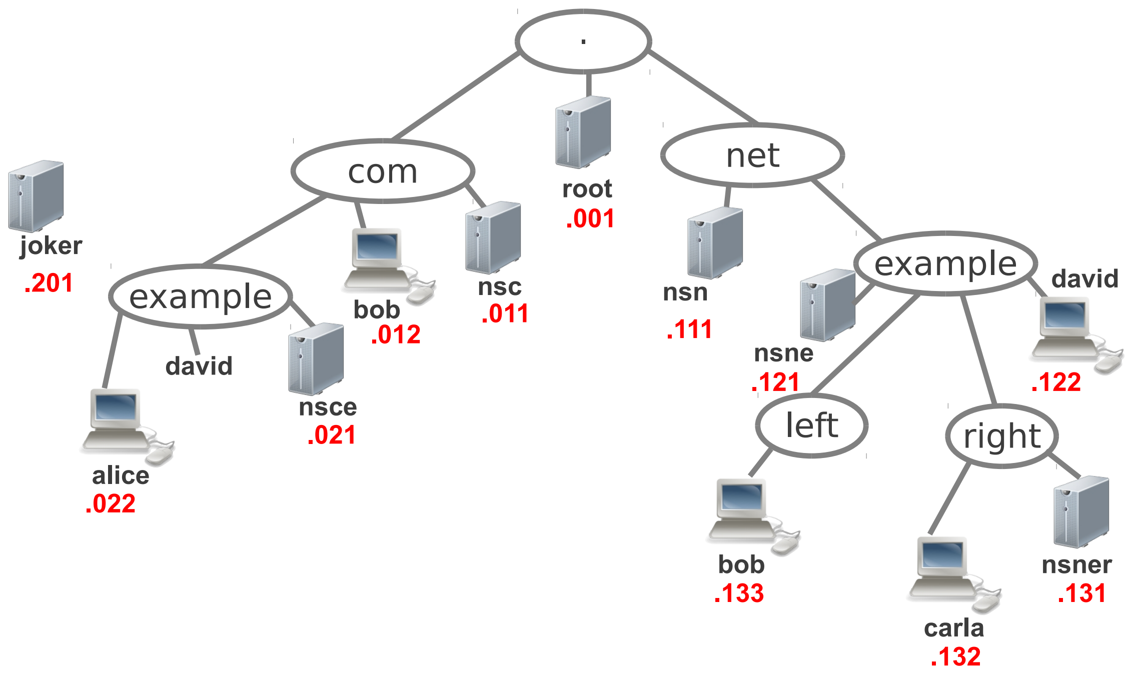


Figure 1.15: Basic network for configuring DNS.

Exercise 1.1– In this exercise, we are going to study the DNS service using the scenario of Figure 1.15. Before the practice you must answer some questions.

To answer consider that ROOT-SERVER is the master of the reverse zone and that it delegates .com to nsc.com and .net to nsn.net. Consider also that nsc.com delegates example.com to nsce.example.com and that nsn.net delegates example.net to nsne.example.net. Finally, consider that nsne.example.net must be configured with a single zone for the zone example.net (single configuration file) and that it must delegate right.example.net to nsner.right.example.net.

According to the previous considerations about our DNS tree, explain in which server we should find the following resource records (RR):

- An A record for peter.example.com
- An A record for peter.left.example.net
- An A record for peter.right.example.net
- The SOA record of right.example.net
- A PTR record for peter.example.com
- A MX record for right.example.net
- A MX record for left.example.net
- A NS record for right.example.net

Now, you can start the scenario *dns-basic* on your host platform:

```
phyhost$ simctl dns-basic start
```

After the scenario has been started, you have to load the initial configuration. Execute the initial label with the following command:

```
phyhost$ simctl dns-basic exec initial
```

Note. You can **reset all the name servers** processes to clear the caches and reload the configuration with the command:

```
phyhost$ simctl dns-basic exec resetbind
```

1. Get a console at **alice** and looking at the configuration explain which is the name server used by this host.

2. Identify which is the server of the zone example.com and describe the configuration of this zone.

3. In **nsce** using the command `netstat`, identify the name of the DNS server process.

4. In this exercise, we analyze a simple query from **alice**. In first place, reset the name servers processes and then, capture with `wireshark tap0` and explain the output of the following command:

   ```
   alice:~# dig alice.example.com
   ```

   Explain also the DNS messages captured with `wireshark`.

   Note. Recall that with the command `dig` you can do DNS queries and show the corresponding responses.

5. Using `dig`, try to resolve the IP address of joker.example.com. Did you find any resolution for this name? Discuss the results.

6. Add the adequate RR in the appropriate server to map the name joker.example.com to the IP address 10.0.0.201. Reset the name server to load the configuration and explain how you test your configuration with `dig` and `ping`.

7. Which IP address will be contacted by a mail server if it has to send an e-mail to john@example.com.

8. Try the following command:

   ```
   alice:~# dig -t MX example.com
   ```

   Explain the output of the command.

Exercise 1.2– Using the scenario *dns-basic*, in this exercise we analyze the recursive queries and the caching strategy of DNS.

1. In this exercise, we analyze a recursive query from **alice**. To do so, reset the name servers processes of the scenario, capture with `wireshark tap0` and explain the flow of DNS messages captured when executing the following command line:

```
alice:~# dig bob.com
```

2. We analyze DNS caching in this exercise. To do so, reset the name servers processes of the scenario, capture with `wireshark tap0` and explain the flow of DNS messages captured when executing the following command line:

   ```
   alice:~# dig bob.com ; sleep 5 ; dig bob.com
   ```

   Note. The `sleep` command delays for a specified amount of seconds.

3. Continuing with the analysis of DNS caching, reset the name servers processes of the scenario, capture with `wireshark tap0` and explain the flow of DNS messages captured when executing the following command line:

   ```
   alice:~# dig bob.com ; sleep 5 ; dig bob.com ; sleep 30 ; dig bob.com
   ```

4. Continuing with the analysis of DNS caching, reset the name servers processes of the scenario, capture with `wireshark tap0` and explain the flow of DNS messages captured when executing the following command line:

   ```
   alice:~# dig alice.com ; sleep 5 ; dig alice.com
   ```

   Note. Take into account that alice.com is not a FQDN under our DNS tree.

5. Set the negative cache TTL to 10 in the SOA of **nsc**. Reset the name servers processes of the scenario, capture with `wireshark tap0` and explain the flow of DNS messages captured when executing the following command line:

   ```
   alice:~# dig alice.com ; sleep 5 ; dig alice.com ; sleep 10 ; dig alice.com
   ```

Exercise 1.3– (*) Using the scenario *dns-basic*, you have to configure the name servers and zones of .net.

1. In your configuration consider that **nsne** must be configured with a single zone for example.net (single configuration file) and that it must delegate right.example.net to **nsner**. Modify the configuration files of **nsn**, **nsne**, and **nsner** appropriately and describe and test your configuration.

2. Notice that the server *nsn* has a mistake in its initial configuration file, describe this mistake.

3. After you have implemented the configuration, reset `bind` in all the name servers of the scenario, capture with `wireshark tap0` and comment the traffic and the results observed when executing:

   ```
   alice:~# dig david.example.com
   ```

Exercise 1.4– (*) In this exercise we study some more features of the DNS service using the same scenario.

1. Use the machine **joker** as a DNS server just for caching and for making queries to the DNS tree on behalf of its clients. Make the clients **alice** and **carla** point to this server and test your configuration, for example asking for one register from **alice** and then, for the same register from **carla**.

2. Change the configuration to delegate the reverse lookup of all the IP addresses of the scenario to the machine **joker**. Describe how you test that your configuration is correct.