1.5. introduction to dns

dns is a fundamental part of every large computer network. **dns** is used by many network services to translate names into network addresses and to locate services on the network (by name).

Whenever you visit a web site, send an e-mail, log on to Active Directory, play Minecraft, chat, or use VoIP, there will be one or (many) more queries to **dns** services.

Should **dns** fail at your organization, then the whole network will grind to a halt (unless you hardcoded the network addresses).

You will notice that even the largest of organizations benefit greatly from having one **dns** infrastructure. Thus **dns** requires all business units to work together.

Even at home, most home modems and routers have builtin **dns** functionality.

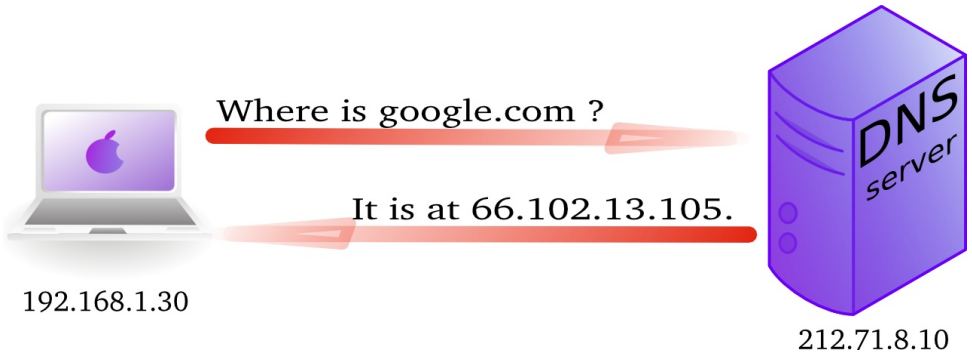
This module will explain what **dns** actually is and how to set it up using **Linux** and **bind9**.

1.5.1. about dns

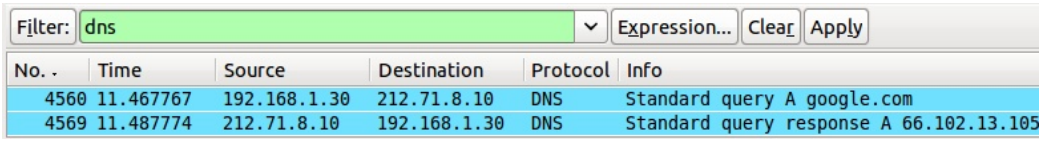
name to ip address resolution

The **domain name system** or **dns** is a service on a tcp/ip network that enables clients to translate names into ip addresses. Actually **dns** is much more than that, but let's keep it simple for now.

When you use a browser to go to a website, then you type the name of that website in the url bar. But for your computer to actually communicate with the web server hosting said website, your computer needs the ip address of that web server. That is where **dns** comes in.



In wireshark you can use the **dns** filter to see this traffic.



history

In the Seventies, only a few hundred computers were connected to the internet. To resolve names, computers had a flat file that contained a table to resolve hostnames to ip addresses. This local file was downloaded from **hosts.txt** on an ftp server in Stanford.

In 1984 **Paul Mockapetris** created **dns**, a distributed treelike hierarchical database that will be explained in detail in these chapters.

Today, **dns** or **domain name system** is a worldwide distributed hierarchical database controlled by **ICANN**. Its primary function is to resolve names to ip addresses, and to point to internet servers providing **smtp** or **ldap** services.

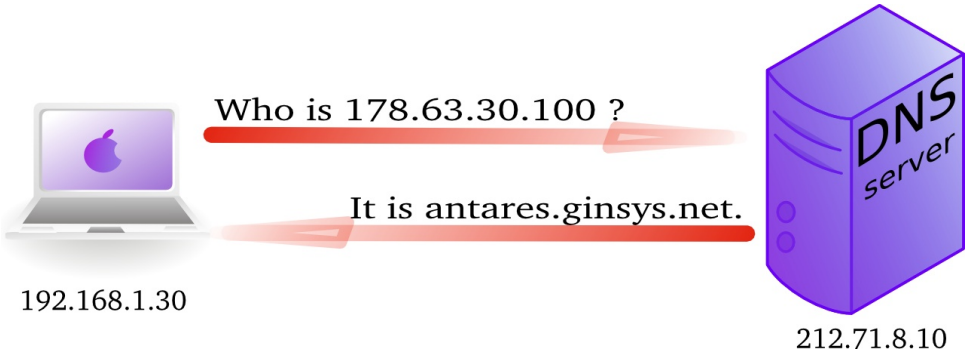
The old **hosts.txt** file is still active today on most computer systems under the name **/etc/hosts** (or C:/Windows/System32/Drivers/etc/hosts). We will discuss this file later, as it can influence name resolution.

forward and reverse lookup queries

The question a client asks a dns server is called a **query**. When a client queries for an ip address, this is called a **forward lookup query** (as seen in the previous drawing).

The reverse, a query for the name of a host, is called a **reverse lookup query**.

Below a picture of a **reverse lookup query**.



Here is a screenshot of a **reverse lookup query** in **nslookup**.

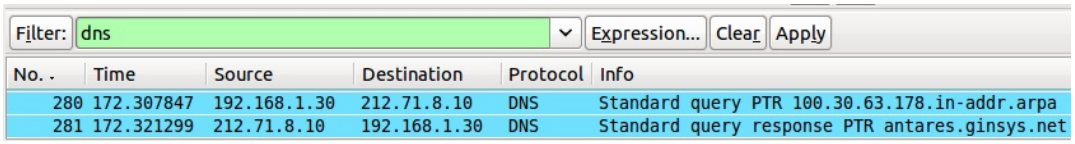
```
root@debian7:~# nslookup
> set type=PTR
> 188.93.155.87
Server:         192.168.1.42
Address:        192.168.1.42#53

Non-authoritative answer:
87.155.93.188.in-addr.arpa      name = antares.ginsys.net.
```

This is what a reverse lookup looks like when sniffing with **tcpdump**.

```
root@debian7:~# tcpdump udp port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
11:01:29.357685 IP 192.168.1.103.42041 > 192.168.1.42.domain: 14763+ PT\
R? 87.155.93.188.in-addr.arpa. (44)
11:01:29.640093 IP 192.168.1.42.domain > 192.168.1.103.42041: 14763 1/0\
/0 PTR antares.ginsys.net. (76)
```

And here is what it looks like in **wireshark** (note this is an older screenshot).



/etc/resolv.conf

A client computer needs to know the ip address of the **dns server** to be able to send queries to it. This is either provided by a **dhcp server** or manually entered.

Linux clients keep this information in the **/etc/resolv.conf** file.

```
root@debian7:~# cat /etc/resolv.conf
domain linux-training.be
search linux-training.be
nameserver 192.168.1.42
root@debian7:~#
```

You can manually change the ip address in this file to use another **dns** server. For example Google provides a public name server at 8.8.8.8 and 8.8.4.4.

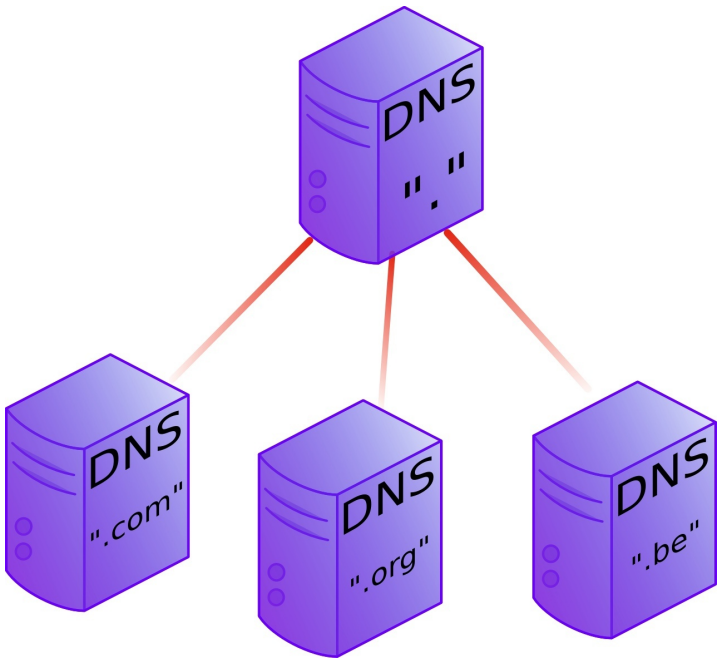
```
root@debian7:~# cat /etc/resolv.conf
nameserver 8.8.8.8
root@debian7:~#
```

Please note that on **dhcp clients** this value can be overwritten when the **dhcp lease** is renewed.

1.5.2. dns namespace

hierarchy

The **dns namespace** is hierarchical tree structure, with the **root servers** (aka dot-servers) at the top. The **root servers** are usually represented by a dot.



Below the **root-servers** are the **Top Level Domains** or **tld**'s.

There are more **tld**'s than shown in the picture. Currently about 200 countries have a **tld**. And there are several general **tld**'s like .com, .edu, .org, .gov, .net, .mil, .int and more recently also .aero, .info, .museum, ...

root servers

There are thirteen **root servers** on the internet, they are named **A** to **M**. Journalists often refer to these servers as **the master servers of the internet**, because if these servers go down, then nobody can (use names to) connect to websites.

The root servers are not thirteen physical machines, they are many more. For example the **F** root server consists of 46 physical machines that all behave as one (using anycast).

```
http://root-servers.org
http://f.root-servers.org
http://en.wikipedia.org/wiki/Root_nameserver.
```

root hints

Every **dns server software** will come with a list of **root hints** to locate the **root servers**.

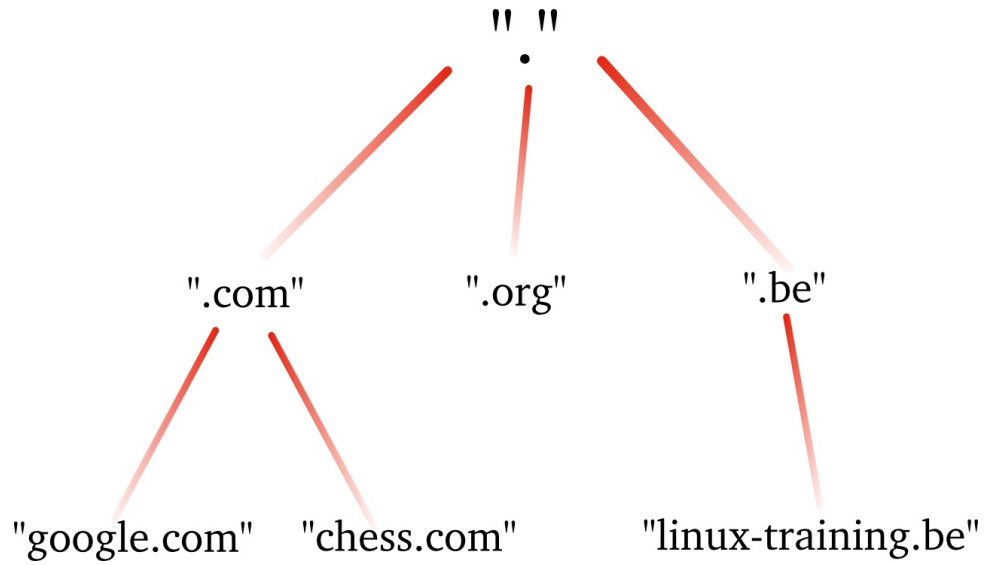
This screenshot shows a small portion of the root hints file that comes with **bind 9.8.4**.

```
root@debian7:~# grep -w 'A ' /etc/bind/db.root
A.ROOT-SERVERS.NET.      3600000      A     198.41.0.4
B.ROOT-SERVERS.NET.      3600000      A     192.228.79.201
C.ROOT-SERVERS.NET.      3600000      A     192.33.4.12
D.ROOT-SERVERS.NET.      3600000      A     199.7.91.13
E.ROOT-SERVERS.NET.      3600000      A     192.203.230.10
F.ROOT-SERVERS.NET.      3600000      A     192.5.5.241
G.ROOT-SERVERS.NET.      3600000      A     192.112.36.4
H.ROOT-SERVERS.NET.      3600000      A     128.63.2.53
I.ROOT-SERVERS.NET.      3600000      A     192.36.148.17
J.ROOT-SERVERS.NET.      3600000      A     192.58.128.30
K.ROOT-SERVERS.NET.      3600000      A     193.0.14.129
L.ROOT-SERVERS.NET.      3600000      A     199.7.83.42
M.ROOT-SERVERS.NET.      3600000      A     202.12.27.33
root@debian7:~#
```

domains

One level below the **top level domains** are the **domains**. Domains can have subdomains (also called child domains).

This picture shows **dns domains** like google.com, chess.com, linux-training.be (there are millions more).



DNS domains are registered at the **tld** servers, the **tld** servers are registered at the **dot servers**.

top level domains

Below the root level are the **top level domains** or **tld's**. Originally there were only seven defined:

Table 1.1. the first top level domains

year	TLD	purpose
1985	.arpa	Reverse lookup via in-addr.arpa
1985	.com	Commercial Organizations
1985	.edu	US Educational Institutions
1985	.gov	US Government Institutions
1985	.mil	US Military
1985	.net	Internet Service Providers, Internet Infrastructure
1985	.org	Non profit Organizations
1988	.int	International Treaties like nato.int

Country **tld**'s were defined for individual countries, like **.uk** in 1985 for Great Britain (yes really), **.be** for Belgium in 1988 and **.fr** for France in 1986. See RFC 1591 for more info.

In 1998 seven new general purpose **tld**'s where chosen, they became active in the 21st century.

Table 1.2. new general purpose tld's

year	TLD	purpose
2002	.aero	aviation related
2001	.biz	businesses
2001	.coop	for co-operatives
2001	.info	informative internet resources
2001	.museum	for museums
2001	.name	for all kinds of names, pseudonyms and labels...
2004	.pro	for professionals

Many people were surprised by the choices, claiming not much use for them and wanting a separate **.xxx** domain (introduced in 2011) for adult content, and **.kidz** a save haven for children. In the meantime more useless **tld**'s were create like **.travel** (for travel agents) and **.tel** (for internet communications) and **.jobs** (for jobs sites).

In 2012 **ICANN** released a list of 2000 new **tld**'s that would gradually become available.

fully qualified domain name

The **fully qualified domain name** or **fqdn** is the combination of the **hostname** of a machine appended with its **domain name**.

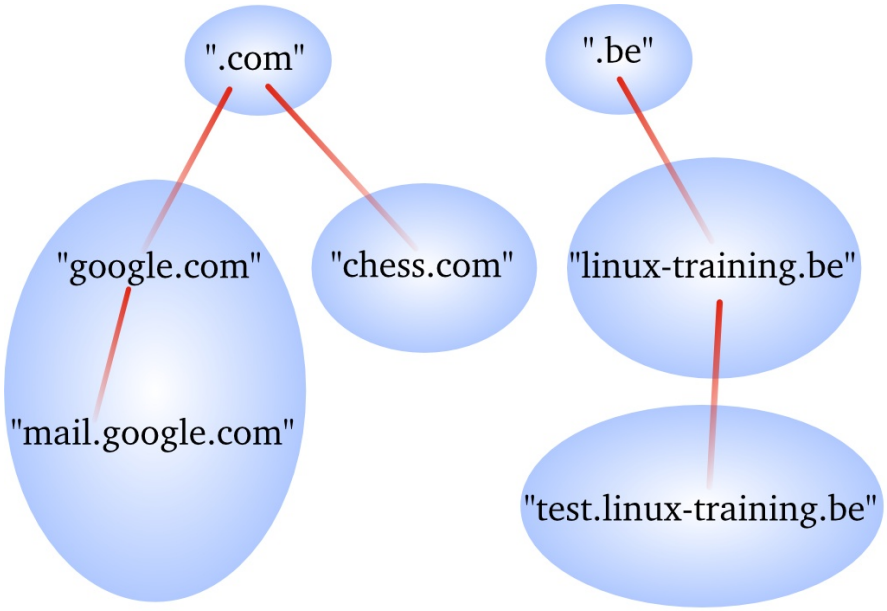
If for example a system is called **gwen** and it is in the domain **linux-training.be**, then the fqdn of this system is **gwen.linux-training.be**.

On Linux systems you can use the **hostname** and **dnsdomainname** commands to verify this information.

```
root@gwen:~# hostname
gwen
root@gwen:~# dnsdomainname
linux-training.be
root@gwen:~# hostname --fqdn
gwen.linux-training.be
root@gwen:~# cat /etc/debian_version
6.0.10
```

dns zones

A **zone** (aka a **zone of authority**) is a portion of the DNS tree that covers one domain name or child domain name. The picture below represents zones as blue ovals. Some zones will contain delegate authority over a child domain to another zone.



A **dns server** can be **authoritative** over 0, 1 or more **dns zones**. We will see more details later on the relation between a **dns server** and a **dns zone**.

A **dns zone** consists of **records**, also called **resource records**. We will list some of those **resource records** on the next page.

dns records

A record

The **A record**, which is also called a **host record** contains the ipv4-address of a computer. When a DNS client queries a DNS server for an A record, then the DNS server will resolve the hostname in the query to an ip address. An **AAAA record** is similar but contains an ipv6 address instead of ipv4.

PTR record

A **PTR record** is the reverse of an A record. It contains the name of a computer and can be used to resolve an ip address to a hostname.

NS record

A **NS record** or **nameserver record** is a record that points to a DNS name server (in this zone). You can list all your name servers for your DNS zone in distinct NS records.

glue A record

An A record that maps the name of an NS record to an ip address is said to be a **glue record**.

SOA record

The SOA record of a zone contains meta information about the zone itself. The contents of the SOA record is explained in detail in the section about zone transfers. There is exactly one SOA record for each zone.

CNAME record

A **CNAME record** maps a hostname to a hostname, creating effectively an alias for an existing hostname. The name of the mail server is often aliased to **mail** or **smtp**, and the name of a web server to **www**.

MX record

The **MX** record points to an **smtp server**. When you send an email to another domain, then your mail server will need the MX record of the target domain's mail server.

1.5.3. caching only servers

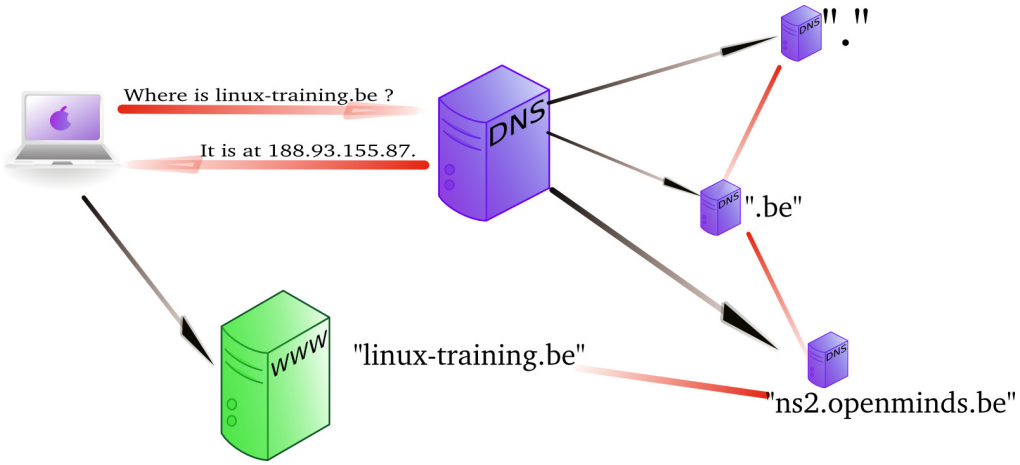
A **dns server** that is set up without **authority** over a **zone**, but that is connected to other name servers and caches the queries is called a **caching only name server**. Caching only name servers do not have a **zone database** with resource records. Instead they connect to other name servers and cache that information.

There are two kinds of caching only name servers. Those with a **forwarder**, and those that use the **root servers**.

caching only server without forwarder

A caching only server without forwarder will have to get information elsewhere. When it receives a query from a client, then it will consult one of the **root servers**. The **root server** will refer it to a **tld** server, which will refer it to another **dns** server. That last server might know the answer to the query, or may refer to yet another server. In the end, our hard working **dns** server will find an answer and report this back to the client.

In the picture below, the clients asks for the ip address of linux-training.be. Our caching only server will contact the root server, and be refered to the .be server. It will then contact the .be server and be refered to one of the name servers of Openminds. One of these name servers (in this cas ns1.openminds.be) will answer the query with the ip address of linux-training.be. When our caching only server reports this to the client, then the client can connect to this website.

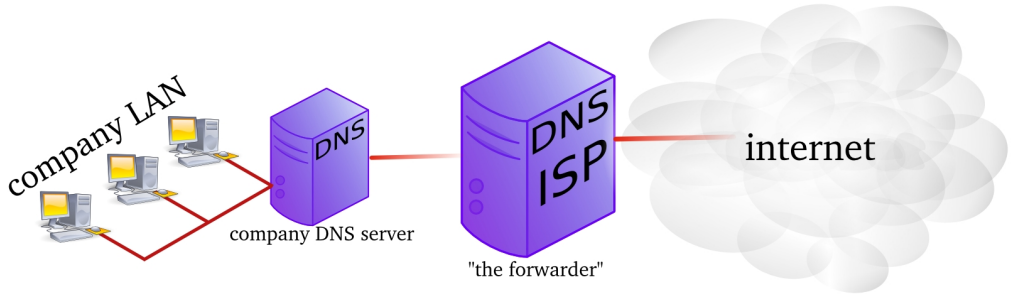


Sniffing with **tcpdump** will give you this (the first 20 characters of each line are cut).

```
192.168.1.103.41251 > M.ROOT-SERVERS.NET.domain: 37279% [1au] A? linux-training.be. (46)
M.ROOT-SERVERS.NET.domain > 192.168.1.103.41251: 37279- 0/11/13 (740)
192.168.1.103.65268 > d.ns.dns.be.domain: 38555% [1au] A? linux-training.be. (46)
d.ns.dns.be.domain > 192.168.1.103.65268: 38555- 0/7/5 (737)
192.168.1.103.7514 > ns2.openminds.be.domain: 60888% [1au] A? linux-training.be. (46)
ns2.openminds.be.domain > 192.168.1.103.7514: 60888*- 1/0/1 A 188.93.155.87 (62)
```

caching only server with forwarder

A **caching only server** with a **forwarder** is a DNS server that will get all its information from the **forwarder**. The **forwarder** must be a **dns server** for example the **dns server** of an **internet service provider**.



This picture shows a **dns server** on the company LAN that has set the **dns server** from their **isp** as a **forwarder**. If the ip address of the **isp dns server** is 212.71.8.10, then the following lines would occur in the **named.conf** file of the company **dns server**:

```
forwarders {
  212.71.8.10;
 };
```

You can also configure your **dns server** to work with **conditional forwarder(s)**. The definition of a conditional forwarder looks like this.

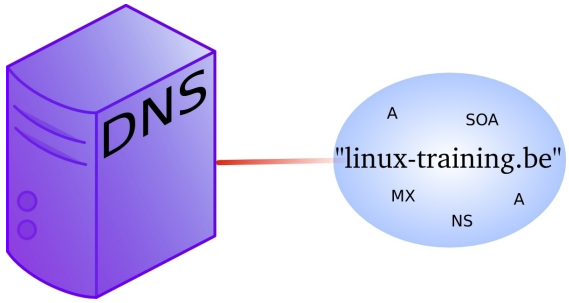
```
zone "someotherdomain.local" {
       type forward;
       forward only;
       forwarders { 10.104.42.1; };
};
```

iterative or recursive query

A **recursive query** is a DNS query where the client that is submitting the query expects a complete answer (Like the fat red arrow above going from the Macbook to the DNS server). An **iterative query** is a DNS query where the client does not expect a complete answer (the three black arrows originating from the DNS server in the picture above). Iterative queries usually take place between name servers. The root name servers do not respond to recursive queries.

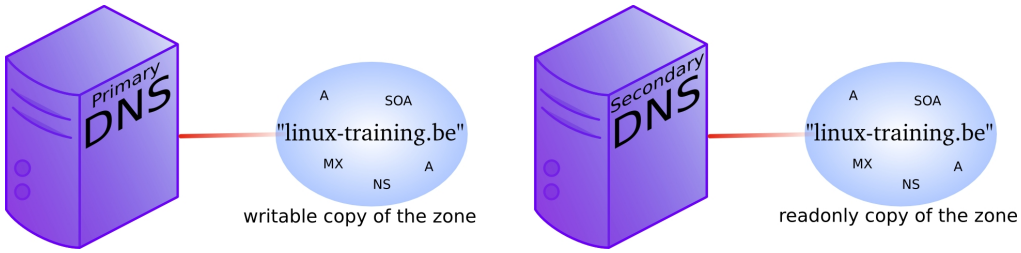
1.5.4. authoritative dns servers

A DNS server that is controlling a zone, is said to be the **authoritative** DNS server for that zone. Remember that a **zone** is a collection of **resource records**.



1.5.5. primary and secondary

When you set up the first **authoritative** dns server for a zone, then this is called the **primary dns server**. This server will have a readable and writable copy of the **zone database**. For reasons of fault tolerance, performance or load balancing you may decide to set up another **dns server** with authority over that zone. This is called a **secondary** dns server.



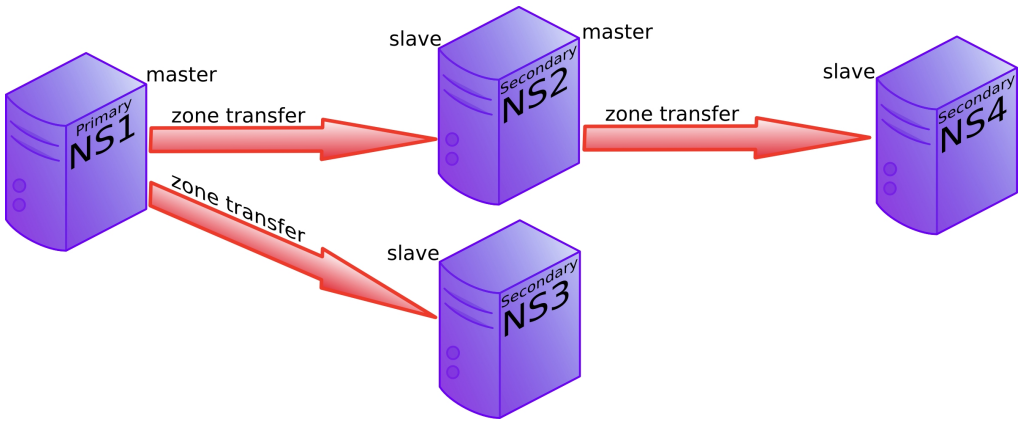
1.5.6. zone transfers

The slave server receives a copy of the zone database from the master server using a **zone transfer**. Zone transfers are requested by the slave servers at regular intervals. Those intervals are defined in the **soa record**.

1.5.7. master and slave

When adding a **secondary dns server** to a zone, then you will configure this server as a **slave server** to the **primary server**. The primary server then becomes the **master server** of the slave server.

Often the **primary dns server** is the **master** server of all slaves. Sometimes a **slave server** is **master server** for a second line slave server. In the picture below ns1 is the primary dns server and ns2, ns3 and ns4 are secondaries. The master for slaves ns2 and ns3 is ns1, but the master for ns4 is ns2.



1.5.8. SOA record

The **soa record** contains a **refresh** value. If this is set to 30 minutes, then the slave server will request a copy of the zone file every 30 minutes. There is also a **retry** value. The retry value is used when the master server did not reply to the last zone transfer request. The value for **expiry time** says how long the slave server will answer to queries, without receiving a zone update.

Below an example of how to use nslookup to query the **soa record** of a zone (linux-training.be).

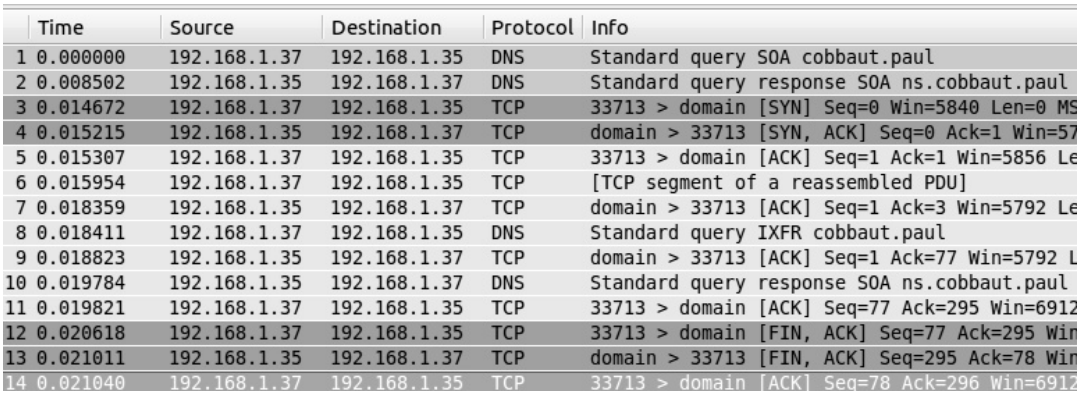
```
root@debian6:~# nslookup
> set type=SOA
> server ns1.openminds.be
> linux-training.be
Server:         ns1.openminds.be
Address:        195.47.215.14#53

linux-training.be
        origin = ns1.openminds.be
        mail addr = hostmaster.openminds.be
        serial = 2321001133
        refresh = 14400
        retry = 3600
        expire = 604800
        minimum = 3600
```

Zone transfers only occur when the zone database was updated (meaning when one or more resource records were added, removed or changed on the master server). The slave server

will compare the **serial number** of its own copy of the SOA record with the serial number of its master's SOA record. When both serial numbers are the same, then no update is needed (because no records were added, removed or deleted). When the slave has a lower serial number than its master, then a zone transfer is requested.

Below a zone transfer captured in wireshark.



	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.37	192.168.1.35	DNS	Standard query SOA cobbaut.paul
2	0.008502	192.168.1.35	192.168.1.37	DNS	Standard query response SOA ns.cobbaut.paul
3	0.014672	192.168.1.37	192.168.1.35	TCP	33713 > domain [SYN] Seq=0 Win=5840 Len=0 MS
4	0.015215	192.168.1.35	192.168.1.37	TCP	domain > 33713 [SYN, ACK] Seq=0 Ack=1 Win=57
5	0.015307	192.168.1.37	192.168.1.35	TCP	33713 > domain [ACK] Seq=1 Ack=1 Win=5856 Le
6	0.015954	192.168.1.37	192.168.1.35	TCP	[TCP segment of a reassembled PDU]
7	0.018359	192.168.1.35	192.168.1.37	TCP	domain > 33713 [ACK] Seq=1 Ack=3 Win=5792 Le
8	0.018411	192.168.1.37	192.168.1.35	DNS	Standard query IXFR cobbaut.paul
9	0.018823	192.168.1.35	192.168.1.37	TCP	domain > 33713 [ACK] Seq=1 Ack=77 Win=5792 L
10	0.019784	192.168.1.35	192.168.1.37	DNS	Standard query response SOA ns.cobbaut.paul
11	0.019821	192.168.1.37	192.168.1.35	TCP	33713 > domain [ACK] Seq=77 Ack=295 Win=6912
12	0.020618	192.168.1.37	192.168.1.35	TCP	33713 > domain [FIN, ACK] Seq=77 Ack=295 Win
13	0.021011	192.168.1.35	192.168.1.37	TCP	domain > 33713 [FIN, ACK] Seq=295 Ack=78 Win
14	0.021040	192.168.1.37	192.168.1.35	TCP	33713 > domain [ACK] Seq=78 Ack=296 Win=6912

1.5.9. full or incremental zone transfers

When a zone tranfer occurs, this can be either a full zone transfer or an incremental zone transfer. The decision depends on the size of the transfer that is needed to completely update the zone on the slave server. An incremental zone transfer is prefered when the total size of changes is smaller than the size of the zone database. Full zone transfers use the **axfr** protocol, incremental zone transfer use the **ixfr** protocol.

1.5.10. DNS cache

DNS is a caching protocol.

When a client queries its local DNS server, and the local DNS server is not authoritative for the query, then this server will go looking for an authoritative name server in the DNS tree. The local name server will first query a root server, then a **tld** server and then a domain server. When the local name server resolves the query, then it will relay this information to the client that submitted the query, and it will also keep a copy of these queries in its cache. So when a(nother) client submits the same query to this name server, then it will retrieve this information form its cache.

For example, a client queries for the A record on www.linux-training.be to its local server. This is the first query ever received by this local server. The local server checks that it is not authoritative for the linux-training.be domain, nor for the **.be tld**, and it is also not a root server. So the local server will use the root hints to send an **iterative** query to a root server.

The root server will reply with a reference to the server that is authoritative for the .be domain (root DNS servers do not resolve fqdn's, and root servers do not respond to recursive queries).

The local server will then sent an iterative query to the authoritative server for the **.be tld**. This server will respond with a reference to the name server that is authoritative for the linux-training.be domain.

The local server will then sent the query for www.linux-training.be to the authoritative server (or one of its slave servers) for the linux-training.be domain. When the local server receives the ip address for www.linux-training.be, then it will provide this information to the client that submitted this query.

Besides caching the A record for www.linux-training.be, the local server will also cache the NS and A record for the linux-training.be name server and the .be name server.

1.5.11. forward lookup zone example

The way to set up zones in **/etc/named.conf.local** is to create a zone entry with a reference to another file (this other file contains the **zone database**).

Here is an example of such an entry in **/etc/named.conf.local**:

```
root@debian7:~# cat /etc/bind/named.conf.local
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "paul.local" IN {
        type master;
        file "/etc/bind/db.paul.local";
        allow-update { none; };
};
root@debian7:~#
```

To create the zone file, the easy method is to copy an existing zone file (this is easier than writing from scratch).

```
root@debian7:/etc/bind# cp db.empty db.paul.local
root@debian7:/etc/bind# vi db.paul.local
```

Here is an example of a zone file.

```
root@debian7:/etc/bind# cat db.paul.local
; zone for classroom teaching
$TTL    86400
@       IN      SOA     debianpaul.paul.local. root.paul.local (
                        2014100100      ; Serial
                        1h              ; Refresh
                        1h              ; Retry
                        2h              ; Expire
                        86400 )         ; Negative Cache TTL
;
; name servers
;
        IN      NS      ns1
        IN      NS      debianpaul
        IN      NS      debian7
;
; servers
;
debianpaul      IN      A       10.104.33.30
debian7         IN      A       10.104.33.30
ns1             IN      A       10.104.33.30
;www            IN      A       10.104.33.30
```

1.5.12. practice: caching only DNS server

1. installing DNS software on Debian

```
root@debian7:~# aptitude update && aptitude upgrade
...
root@debian7:~# aptitude install bind9
...
root@debian7:~# dpkg -l | grep bind9 | tr -s ' '
ii bind9 1:9.8.4.dfsg.P1-6+nmu2+deb7u2 amd64 Internet Domain Name Server
ii bind9-host 1:9.8.4.dfsg.P1-6+nmu2+deb7u2 amd64 Version of 'host' bundled with BIND 9.X
ii bind9utils 1:9.8.4.dfsg.P1-6+nmu2+deb7u2 amd64 Utilities for BIND
ii libbind9-80 1:9.8.4.dfsg.P1-6+nmu2+deb7u2 amd64 BIND9 Shared Library used by BIND
root@debian7:~#
```

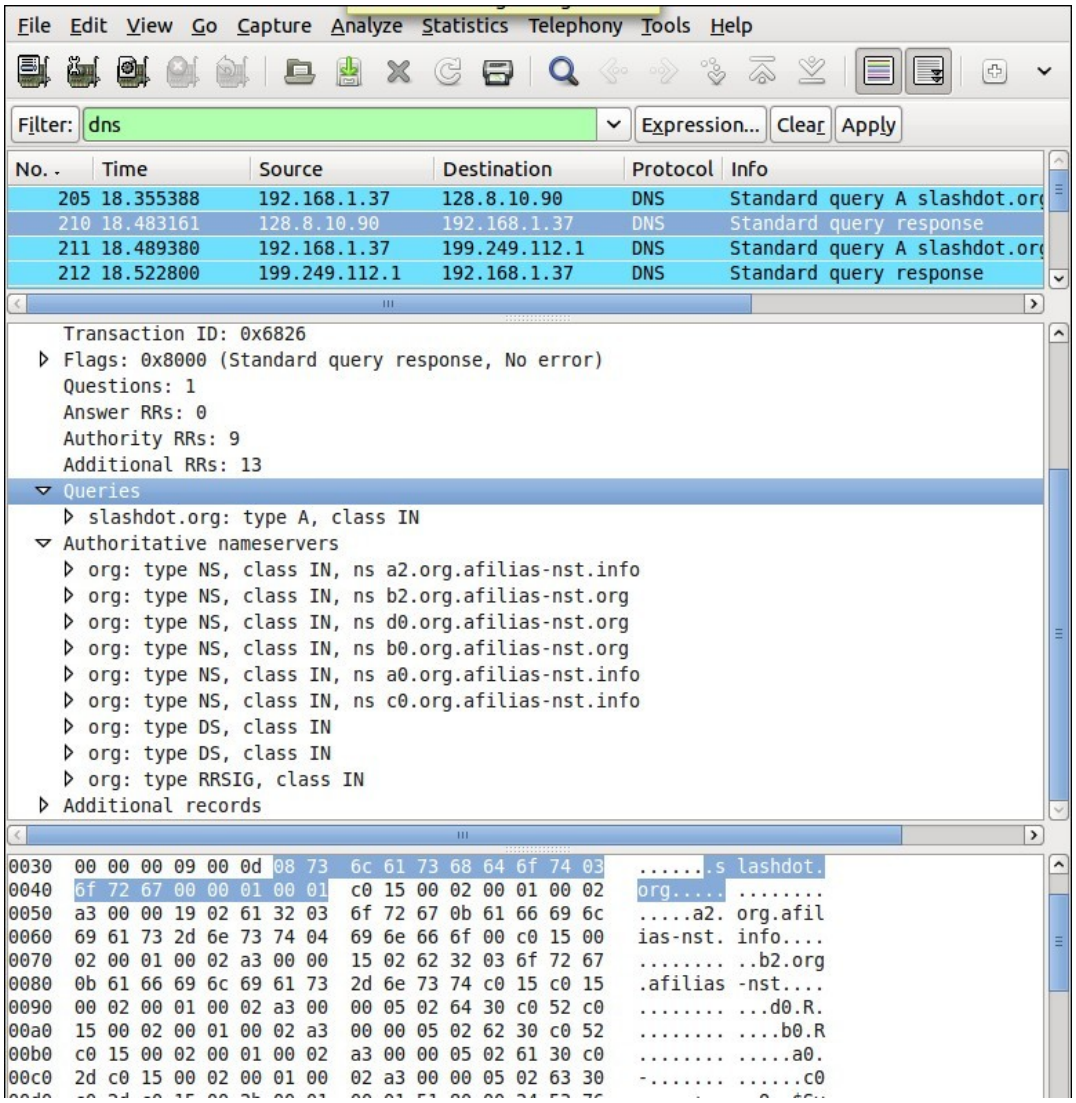
2. Discover the default configuration files. Can you define the purpose of each file ?

```
root@debian7:~# ls -l /etc/bind
total 52
-rw-r--r-- 1 root root 2389 Sep  5 20:25 bind.keys
-rw-r--r-- 1 root root  237 Sep  5 20:25 db.0
-rw-r--r-- 1 root root  271 Sep  5 20:25 db.127
-rw-r--r-- 1 root root  237 Sep  5 20:25 db.255
-rw-r--r-- 1 root root  353 Sep  5 20:25 db.empty
-rw-r--r-- 1 root root  270 Sep  5 20:25 db.local
-rw-r--r-- 1 root root 3048 Sep  5 20:25 db.root
-rw-r--r-- 1 root bind  463 Sep  5 20:25 named.conf
-rw-r--r-- 1 root bind  490 Sep  5 20:25 named.conf.default-zones
-rw-r--r-- 1 root bind  374 Oct  1 20:01 named.conf.local
-rw-r--r-- 1 root bind  913 Oct  1 13:24 named.conf.options
-rw-r----- 1 bind bind   77 Oct  1 11:14 rndc.key
-rw-r--r-- 1 root root 1317 Sep  5 20:25 zones.rfc191
```

3. Setup caching only dns server. This is normally the default setup. A caching-only name server will look up names for you and cache them. Many tutorials will tell you to add a **forwarder**, but we first try without this!

Hey this seems to work without a **forwarder**. Using a sniffer you can find out what really happens. Your freshly install dns server is not using a cache, and it is not using your local dns server (from /etc/resolv.conf). So where is this information coming from ? And what can you learn from sniffing this dns traffic ?

4. Explain in detail what happens when you enable a caching only dns server without forwarder. This wireshark screenshot can help, but you learn more by sniffing the traffic yourself.



You should see traffic to a **root name server** whenever you try a new **tld** for the first time. Remember that **dns** is a caching protocol, which means that repeating a query will generate a lot less traffic since your **dns server** will still have the answer in its memory.

1.5.13. practice: caching only with forwarder

5. Add the public Google **dns server** as a **forwarder**. The ip address of this server is 8.8.8.8 .

Before the change:

```
root@debian7:~# grep -A2 'forwarders {' /etc/bind/named.conf.options
        // forwarders {
        //      0.0.0.0;
        // };
```

changing:

```
root@debian7:~# vi /etc/bind/named.conf.options
```

After the change:

```
root@debian7:~# grep -A2 'forwarders {' /etc/bind/named.conf.options
         forwarders {
                8.8.8.8;
         };
root@debian7:~#
```

Restart the server:

```
root@debian7:~# service bind9 restart
Stopping domain name service...: bind9.
Starting domain name service...: bind9.
```

6. Explain the purpose of adding the **forwarder**. What is our **dns server** doing when it receives a query ?

```
root@debian7:~# nslookup
> server
Default server: 10.104.33.30
Address: 10.104.33.30#53
> linux-training.be
Server:         10.104.33.30
Address:        10.104.33.30#53

Non-authoritative answer:
Name:   linux-training.be
Address: 188.93.155.87
>
```

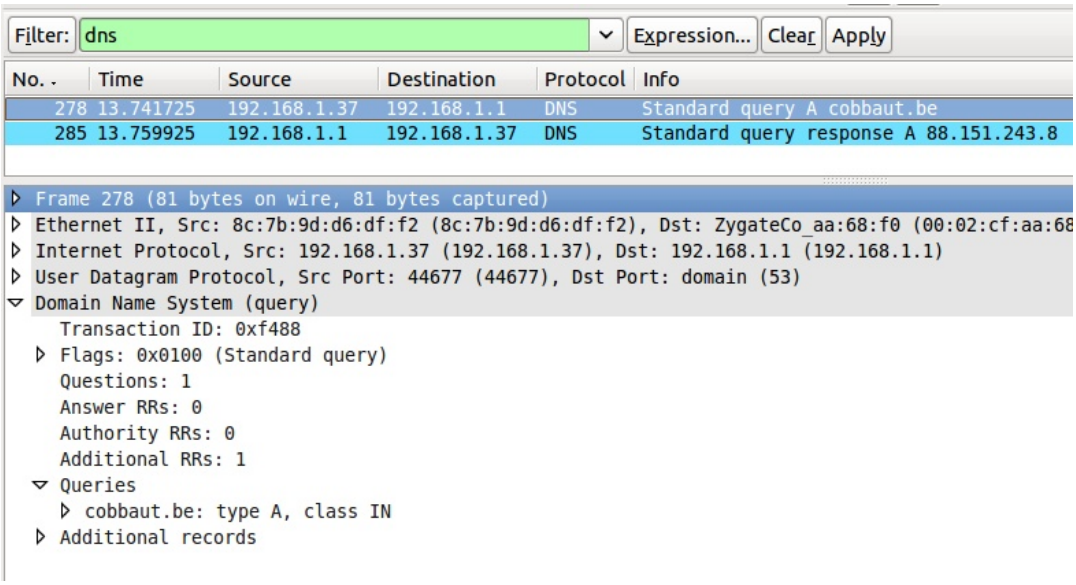
This is the output of **tcpdump udp port 53** while executing the above query for **linux-trainign.be** in **nslookup**.

```
root@debian7:~# tcpdump udp port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

You should find the following two lines in the output of **tcpdump**:

```
10.104.33.30.19381 > google-public-dns-a.google.com.domain: 18237+% [1au] A? linux-training.be
google-public-dns-a.google.com.domain > 10.104.33.30.19381: 18237 1/0/1 A 188.93.155.87 (62)
```

Below is an (old) wireshark screenshot that can help, you should see something similar (but with different ip addresses).



7. What happens when you query for the same domain name more than once ?

8. Why does it say "non-authoritative answer" ? When is a dns server authoritative ?

9. You can also use **dig** instead of **nslookup**.

```
dig @192.168.1.37 linux-training.be
```

10. How can we avoid having to set the server in dig or nslookup ?

Change this:

```
root@debian7:~# cat /etc/resolv.conf
nameserver 10.46.101.1
root@debian7:~#
```

into this:

```
root@debian7:~# cat /etc/resolv.conf
nameserver 10.104.33.30
root@debian7:~#
```

11. When you use **dig** for the first time for a domain, where is the answer coming from ? And the second time ? How can you tell ?

1.5.14. practice: primary authoritative server

1. Instead of only cachng the information from other servers, we will now make our server authoritative for our own domain.

2. I choose the top level domain **.local** and the domain **paul.local** and put the information in **/etc/bind/named.conf.local**.

```
root@debian7:~# cat /etc/bind/named.conf.local
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "paul.local" IN {
        type master;
        file "/etc/bind/db.paul.local";
        allow-update { none; };
};
```

3. Also add a **zone database file**, similar to this one (add some A records for testing). Set the **Refresh** and **Retry** values not too high so you can sniff this traffic (this example makes the slave server contact the master every hour).

```
root@debian7:~# cat /etc/bind/db.paul.local
; zone for classroom teaching
$TTL    86400
@       IN      SOA     debianpaul.paul.local. root.paul.local (
                        2014100101      ; Serial
                        1h              ; Refresh
                        1h              ; Retry
                        2h              ; Expire
                        900 )           ; Negative Cache TTL
;
; name servers
;
        IN      NS      ns1
        IN      NS      debianpaul
        IN      NS      debian7
;
; servers
;
debianpaul      IN      A       10.104.33.30
debian7         IN      A       10.104.33.30
ns1             IN      A       10.104.33.30
;www            IN      A       10.104.33.30
root@debian7:~#
```

Note that the **www** record is commented out, so it will not resolve.

4. Restart the DNS server and check your zone in the error log.

```
root@debian7:~# service bind9 restart
Stopping domain name service...: bind9.
Starting domain name service...: bind9.
root@debian7:~# grep paul.local /var/log/syslog
Oct  6 09:22:18 pauldebian named[2707]: zone paul.local/IN: loaded serial 2014100101
Oct  6 09:22:18 pauldebian named[2707]: zone paul.local/IN: sending notifies (serial 201410010
root@debian7:~#
```

5. Use **dig** or **nslookup** (or even **ping**) to test your A records.

```
root@debian7:~# ping -c1 ns1.paul.local
PING ns1.paul.local (10.104.33.30) 56(84) bytes of data.
64 bytes from 10.104.33.30: icmp_req=1 ttl=64 time=0.006 ms

--- ns1.paul.local ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.006/0.006/0.006/0.000 ms
root@debian7:~# ping -c1 www.paul.local
ping: unknown host www.paul.local
root@debian7:~#
```

Note that the **www** record was commented out, so it should fail.

```
root@debian7:~# dig debian7.paul.local

; <<>> DiG 9.8.4-rpz2+rl005.12-P1 <<>> debian7.paul.local
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50491
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 2

;; QUESTION SECTION:
;debian7.paul.local.            IN      A

;; ANSWER SECTION:
debian7.paul.local.     86400   IN      A       10.104.33.30

;; AUTHORITY SECTION:
paul.local.             86400   IN      NS      ns1.paul.local.
paul.local.             86400   IN      NS      debian7.paul.local.
paul.local.             86400   IN      NS      debianpaul.paul.local.

;; ADDITIONAL SECTION:
ns1.paul.local.         86400   IN      A       10.104.33.30
debianpaul.paul.local.  86400   IN      A       10.104.33.30

;; Query time: 4 msec
;; SERVER: 10.104.33.30#53(10.104.33.30)
;; WHEN: Mon Oct  6 09:35:25 2014
;; MSG SIZE  rcvd: 141

root@debian7:~#
```

6. Our primary server appears to be up and running. Note the information here:

```
server os  : Debian 7
ip address : 10.104.33.30
domain name: paul.local
server name: ns1.paul.local
```

1.5.15. practice: reverse DNS

1. We can add ip to name resolution to our dns-server using a reverse dns zone.

2. Start by adding a .arpa zone to /etc/bind/named.conf.local like this (we set notify to no to avoid sending of notify messages to other name servers):

```
root@ubu1010srv:/etc/bind# grep -A4 arpa named.conf.local
zone "1.168.192.in-addr.arpa" {
 type master;
 notify no;
 file "/etc/bind/db.192";
};
```

3. Also create a zone database file for this reverse lookup zone.

```
root@ubu1010srv:/etc/bind# cat db.192
;
; BIND reverse data file for 192.168.1.0/24 network
;
$TTL 604800
@ IN SOA ns.cobbaut.paul root.cobbaut.paul. (
   20110516 ; Serial
    604800  ; Refresh
     86400  ; Retry
   2419200  ; Expire
    604800 ) ; Negative Cache TTL
;
@ IN NS ns.
37 IN PTR ns.cobbaut.paul.
1 IN PTR anya.cobbaut.paul.
30 IN PTR mac.cobbaut.paul.
root@ubu1010srv:/etc/bind#
```

4. Test with nslookup or dig:

```
root@ubu1010srv:/etc/bind# dig 1.168.192.in-addr.arpa AXFR
```

1.5.16. practice: a DNS slave server

1. A slave server transfers zone information over the network from a master server (a slave can also be a master). A primary server maintains zone records in its local file system. As an exercise, and to verify the work of all students, set up a slave server of all the master servers in the classroom.

2. Before configuring the slave server, we have to allow transfers from our zone to this server. Remember that this is not very secure since transfers are in clear text and limited to an ip address. This example follows our demo from above. The ip of my slave server is 192.168.1.31, yours is probably different.

```
root@ubu1010srv:/etc/bind# grep -A2 cobbaut named.conf.local
zone "paul.local" {
 type master;
 file "/etc/bind/db.paul.local";
 allow-transfer { 192.168.1.31; };
};
```

3. My slave server is running Fedora 14. Bind configuration files are only a little different. Below the addition of a slave zone to this server, note the ip address (192.168.1.37) of my master dns server for the cobbaut.paul zone.

```
[root@fedora14 etc]# grep cobbaut -A2 named.conf
zone "cobbaut.paul" {
 type slave;
 file "/var/named/slaves/db.cobbaut.paul";
 masters { 192.168.1.37; };
};
[root@fedora14 etc]#
```

4. You might need to add the ip address of the server on Fedora to allow queries other than from localhost.

```
[root@fedora14 etc]# grep 127 named.conf
 listen-on port 53 { 127.0.0.1; 192.168.1.31; };
```

5. Restarting bind on the slave server should transfer the zone database file:

```
[root@fedora14 etc]# ls -l /var/named/slaves/
total 4
-rw-r--r--. 1 named named 387 May 16 03:23 db.cobbaut.paul
[root@fedora14 etc]#
```