dns server

Paul Cobbaut

dns server

Paul Cobbaut

lt-1.9

Published Wed 05 Dec 2012 05:25:38 PM CET

Abstract

Dit boek wordt gebruikt als beknopte handleiding voor de opleiding "bachelor informatica" (HBO5 Informatica) in avondschool. **Dit boek bevat niet alle leerstof, vergeet niet om zelf te noteren.**

Meer informatie en een gratis .pdf is beschikbaar op **http://linux-training.be** .

Feel free to contact the authors:

- Paul Cobbaut: paul.cobbaut@gmail.com, http://www.linkedin.com/in/cobbaut

Contributors to the Linux Training project are:

- Serge van Ginderachter: serge@ginsys.be, build scripts; infrastructure setup; minor stuff
- Hendrik De Vloed: hendrik.devloed@ugent.be, buildheader.pl script

Table of Contents

List of Tables

Chapter 1. introduction to DNS

Table of Contents

Every computer on the internet is connected to a huge worldwide tree of **dns** servers. Most organisations have more than one **dns server**, and even Personal Area Networks have a **built-in dns server** in a small modem or router.

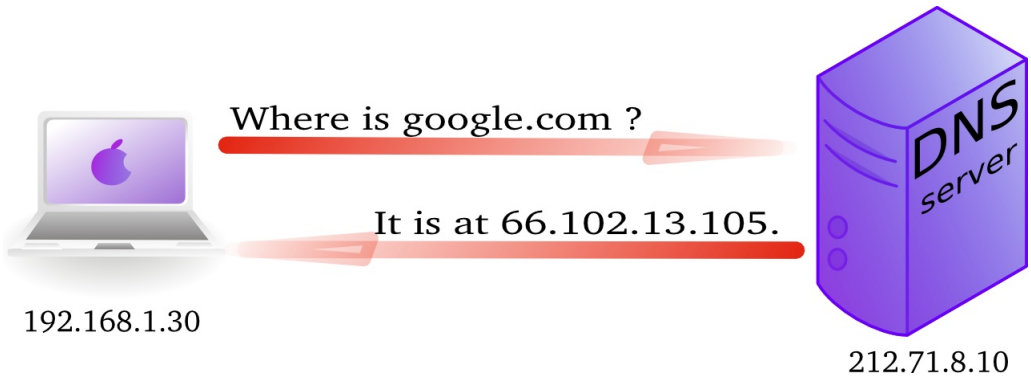
In this chapter we will explain what **dns** actually is and how to set it up using Linux.

1.1. about dns

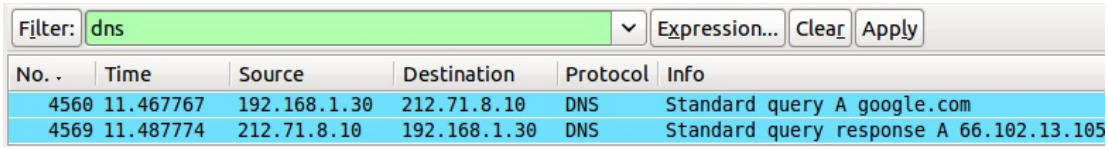
1.1.1. name to ip-address resolution

The **domain name system** or **dns** is a service on a tcp/ip network that enables clients to translate names into ip-addresses. It 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.



1.1.2. 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.

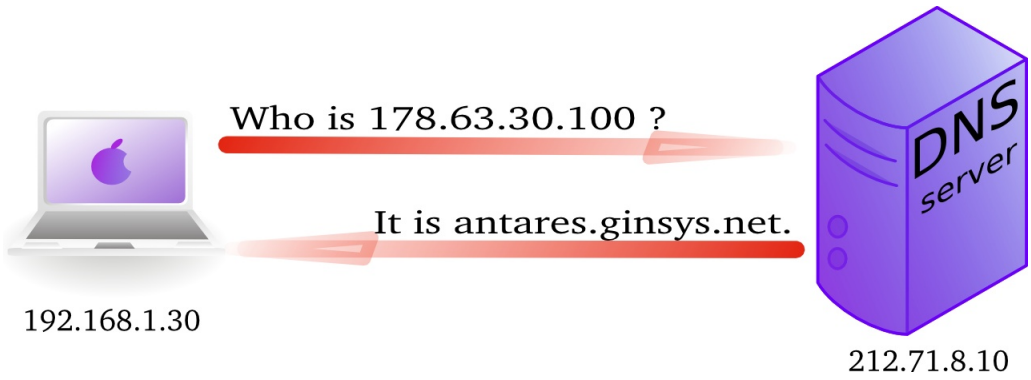
The old **hosts.txt** file is still active today on most computer systems under the name **/etc/hosts**. We will discuss this file later, as it can influence name resolution.

1.1.3. 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**.

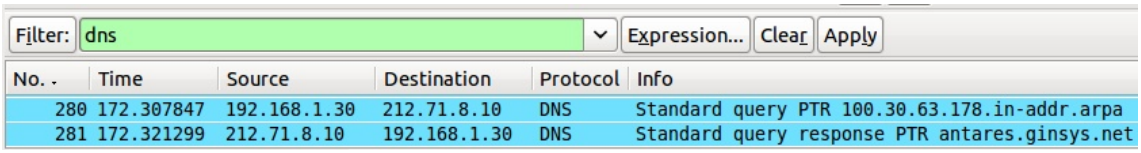
Below a picture of a **reverse lookup query**.



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

```
paul@ubu1010:~$ nslookup
> set type=PTR
> 178.63.30.100
Server:  212.71.8.10
Address: 212.71.8.10#53

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

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



1.1.4. /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.

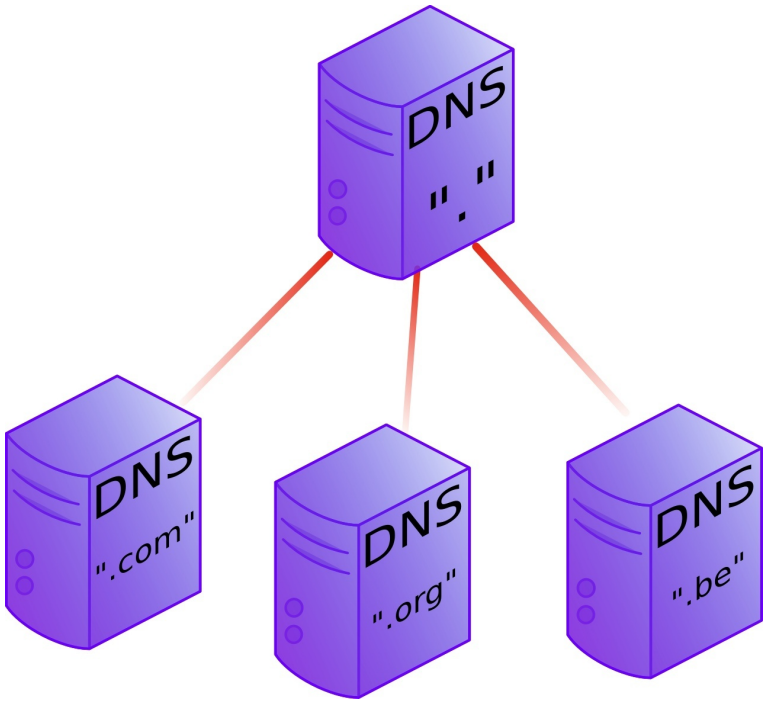
Linux clients keep this information in the **/etc/resolv.conf** file.

```
paul@ubu1010:~$ cat /etc/resolv.conf
nameserver 212.71.8.10
```

1.2. dns namespace

1.2.1. 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, ...

1.2.2. 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.
```

1.2.3. root hints

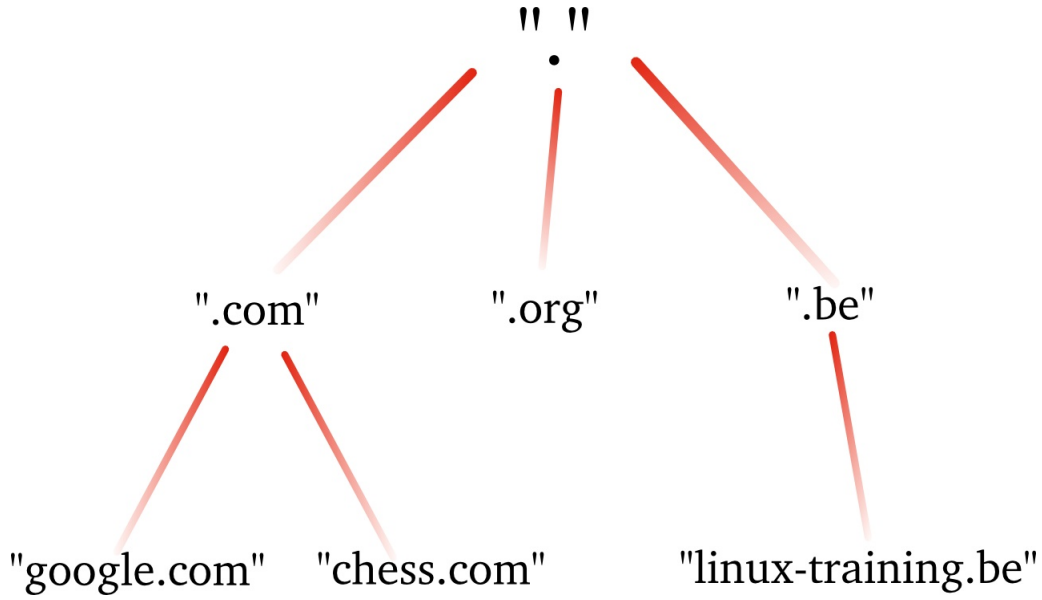
Every **dns server software** will come with a list of **root hints** to locate the **root servers**.

```
root@gwen:~# grep ' 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     128.8.10.90
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
```

1.2.4. 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**.

1.2.5. 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).

1.2.6. 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**.

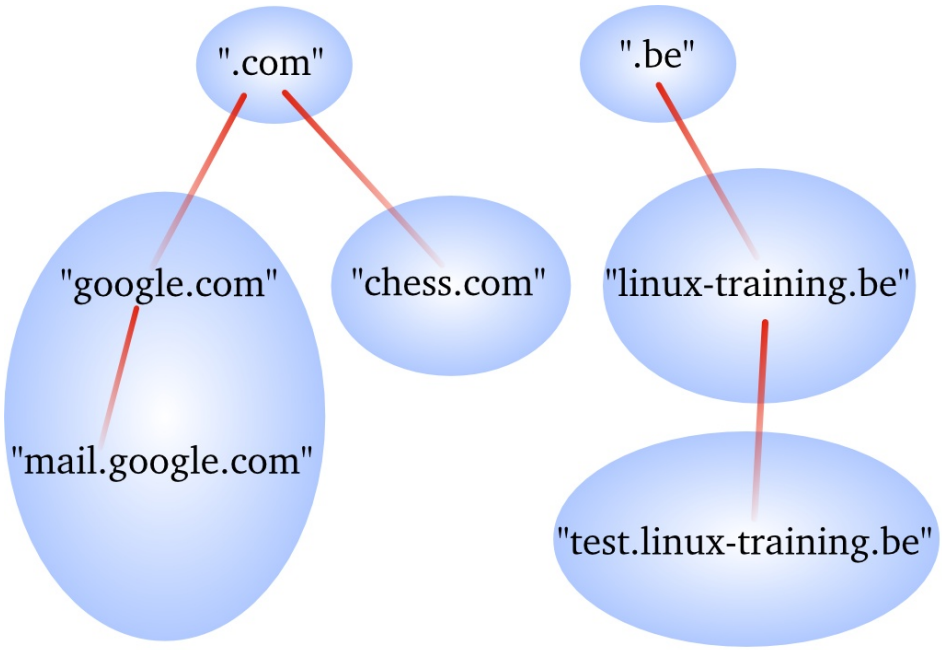
If for example a system is called **gwen** and it is in the domain **linux-training.be**, then the fqdn of this system is **gwen.linux-training.be**.

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

```
root@gwen:~# hostname
gwen
root@gwen:~# domainname
linux-training.be
root@gwen:~# hostname --fqdn
gwen.linux-training.be
```

1.2.7. 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.

1.2.8. 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.

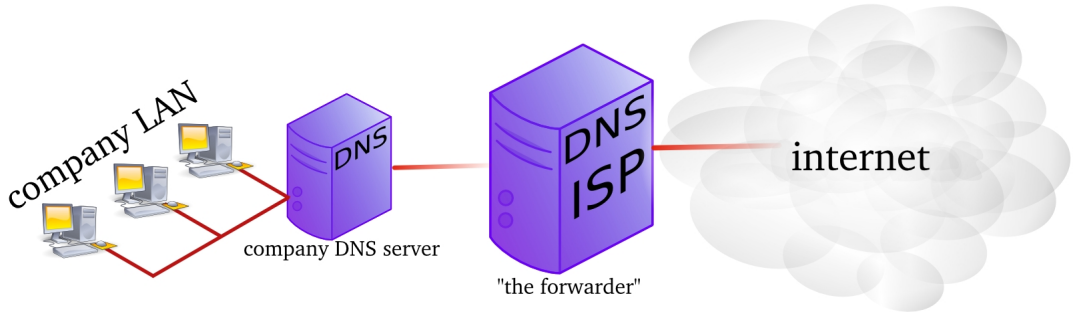
1.3. caching only servers

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**.

1.3.1. 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**.



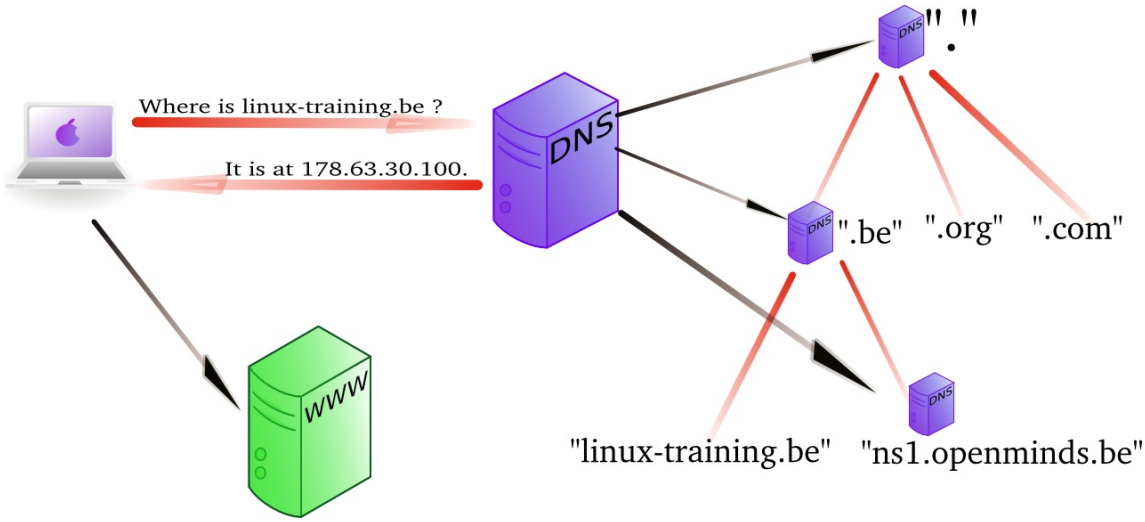
This picture shows a **dns server** on the company LAN that has set the **dns server** from their **isp** as a **forwarder**. If the ip address of the **isp dns server** is 212.71.8.10, then the following lines would occur in the **named.conf** file of the company **dns server**:

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

1.3.2. 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.

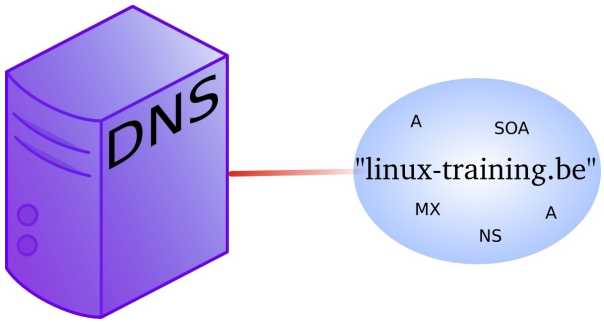


1.3.3. iterative or recursive query

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.

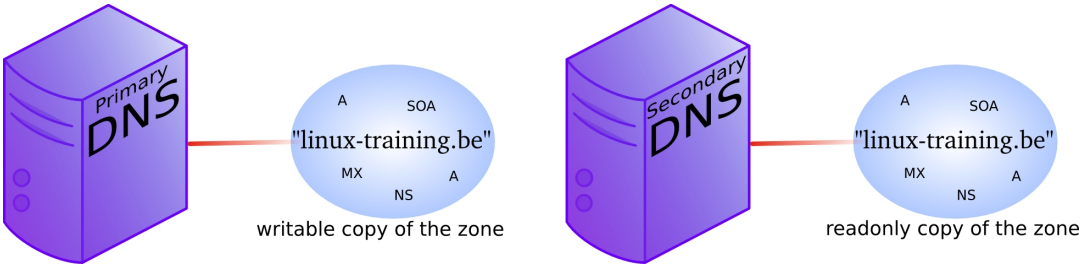
1.4. authoritative dns servers

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. 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.



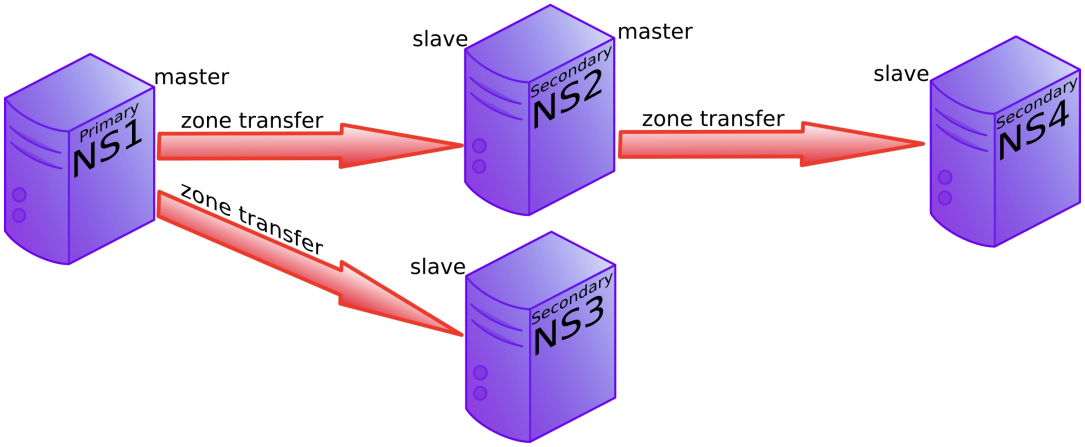
1.6. zone transfers

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**.

1.7. 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.



1.8. 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.

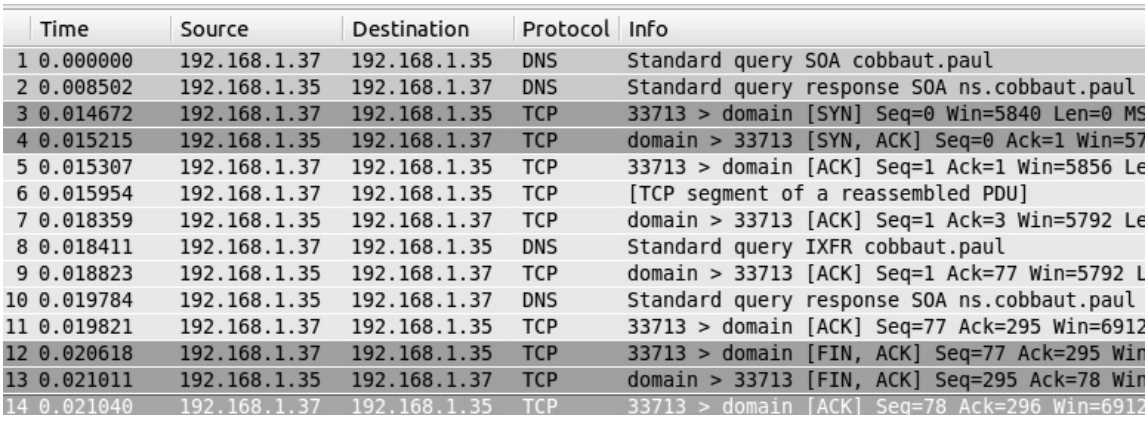
Below an example of how to use nslookup to query the **soa record** of a zone (linux-training.be).

```
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

1.9. 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.

1.10. 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.

1.11. forward lookup zone example

The way to set up zones in **/etc/named.conf** is to create a zone entry with a reference to another file located in /var/named.

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

```
zone "classdemo.local" IN {
 type master;
 file "classdemo.local.zone";
 allow-update { none; };
};
```

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

```
[root@RHEL4b named]# cd /var/named/
[root@RHEL4b named]# pwd
/var/named
[root@RHEL4b named]# cp localhost.zone classdemo.local.zone
[root@RHEL4b named]#
```

Here is an example of a zone file.

```
[root@RHEL4b named]# cat classdemo.local.zone
$TTL    86400
$ORIGIN classdemo.local.
@       IN SOA  rhel4b.classdemo.local.   admin.classdemo.local. (
                          2007083100      ; serial
                          3H              ; refresh
                          900             ; retry
                          1W              ; expiry
                          1D )            ; minimum

                IN NS           rhel4b.classdemo.local.
                IN MX     10    mail.classdemo.local.
                IN A            192.168.1.191

rhel4b          IN      A       192.168.1.191
mail            IN      A       192.168.1.191
www             IN      A       192.168.1.191
ftp             IN      A       192.168.1.191
server2         IN      A       192.168.1.1
```

1.12. Practice: caching only DNS server

1a. installing DNS software on Debian/Ubuntu

```
root@ubu1010srv:~# dpkg -l | grep bind9
ii   bind9-host    1:9.7.1.dfsg.P2-2ubuntu0.2   Version of 'host' bun\
dled with BIND 9.X
ii   libbind9-60   1:9.7.1.dfsg.P2-2ubuntu0.2   BIND9 Shared Library \
used by BIND
root@ubu1010srv:~# aptitude install bind9
The following NEW packages will be installed:
  bind9 bind9utils{a}
0 packages upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 433kB of archives. After unpacking 1,352kB will be used.
Do you want to continue? [Y/n/?]

... output truncated ...

 * Starting domain name service... bind9                        [ OK ]

root@ubu1010srv:~# dpkg -l | grep bind9
ii bind9       1:9.7.1.dfsg.P2-2ubuntu0.2 Internet Domain Name Server
ii bind9-host  1:9.7.1.dfsg.P2-2ubuntu0.2 Version of 'host' bundled w\
ith BIND 9.X
ii bind9utils  1:9.7.1.dfsg.P2-2ubuntu0.2 Utilities for BIND
ii libbind9-60 1:9.7.1.dfsg.P2-2ubuntu0.2 BIND9 Shared Library used b\
y BIND
root@ubu1010srv:~#
```

1b. installing DNS software on RHEL/Fedora

```
[root@fedora14 ~]# rpm -qa | grep bind
samba-winbind-clients-3.5.8-74.fc14.i686
bind-utils-9.7.3-1.fc14.i686
PackageKit-device-rebind-0.6.12-2.fc14.i686
bind-libs-9.7.3-1.fc14.i686
[root@fedora14 ~]# yum install bind
Loaded plugins: langpacks, presto, refresh-packagekit
Adding en_US to language list
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package bind.i686 32:9.7.3-1.fc14 set to be installed
--> Finished Dependency Resolution

...output truncated

Running Transaction
  Installing     : 32:bind-9.7.3-1.fc14.i686                   1/1

Installed:
  bind.i686 32:9.7.3-1.fc14

Complete!
[root@fedora14 ~]# rpm -qa | grep bind
samba-winbind-clients-3.5.8-74.fc14.i686
bind-utils-9.7.3-1.fc14.i686
PackageKit-device-rebind-0.6.12-2.fc14.i686
bind-libs-9.7.3-1.fc14.i686
bind-9.7.3-1.fc14.i686
[root@fedora14 ~]#
```

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

2a. On Fedora:

```
[root@fedora14 ~]# ls -ld /etc/named*
drwxr-x---. 2 root named 4096 Feb 18 16:07 /etc/named
-rw-r-----. 1 root named 1008 Jul 19  2010 /etc/named.conf
-rw-r--r--. 1 root named 2544 Feb 18 16:07 /etc/named.iscdlv.key
-rw-r-----. 1 root named  931 Jun 21  2007 /etc/named.rfc1912.zones
-rw-r--r--. 1 root named  487 Jul 19  2010 /etc/named.root.key
[root@fedora14 ~]# ls -l /var/named/
total 28
drwxrwx---. 2 named named 4096 Feb 18 16:07 data
drwxrwx---. 2 named named 4096 Feb 18 16:07 dynamic
-rw-r-----. 1 root  named 1892 Feb 18  2008 named.ca
-rw-r-----. 1 root  named  152 Dec 15  2009 named.empty
-rw-r-----. 1 root  named  152 Jun 21  2007 named.localhost
-rw-r-----. 1 root  named  168 Dec 15  2009 named.loopback
drwxrwx---. 2 named named 4096 Feb 18 16:07 slaves
```

2. On Ubuntu:

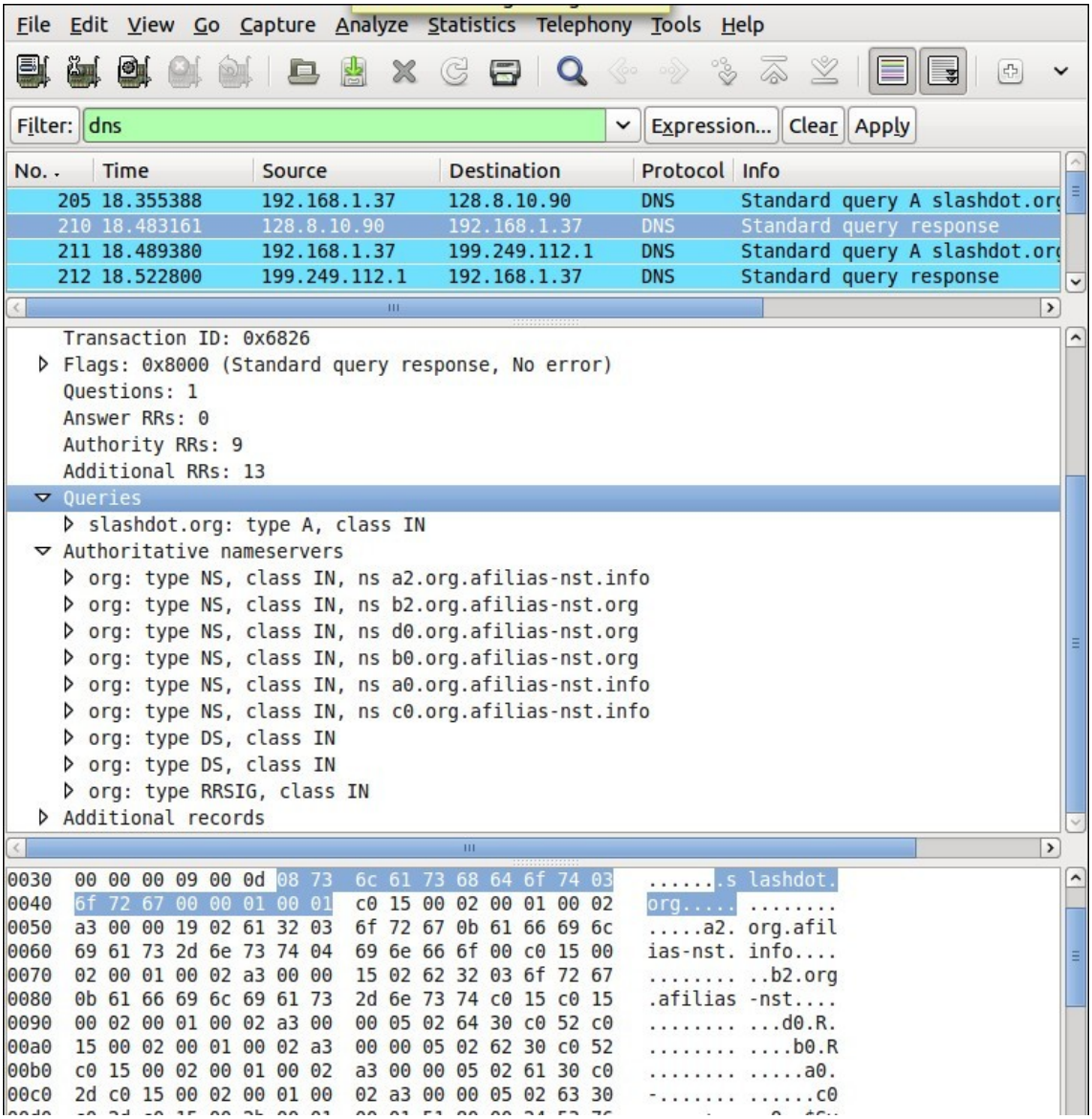
```
root@ubu1010srv:~# ls -l /etc/bind
total 52
-rw-r--r-- 1 root root  601 2011-02-23 16:22 bind.keys
-rw-r--r-- 1 root root  237 2011-02-23 16:22 db.0
-rw-r--r-- 1 root root  271 2011-02-23 16:22 db.127
-rw-r--r-- 1 root root  237 2011-02-23 16:22 db.255
-rw-r--r-- 1 root root  353 2011-02-23 16:22 db.empty
-rw-r--r-- 1 root root  270 2011-02-23 16:22 db.local
-rw-r--r-- 1 root root 2994 2011-02-23 16:22 db.root
-rw-r--r-- 1 root bind  463 2011-02-23 16:22 named.conf
-rw-r--r-- 1 root bind  490 2011-02-23 16:22 named.conf.default-zones
-rw-r--r-- 1 root bind  165 2011-02-23 16:22 named.conf.local
-rw-r--r-- 1 root bind  572 2011-02-23 16:22 named.conf.options
-rw-r----- 1 bind bind   77 2011-05-15 17:52 rndc.key
-rw-r--r-- 1 root root 1317 2011-02-23 16:22 zones.rfc1918
```

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. Most tutorials will tell you to add a **forwarder**, so we first try without this!

```
root@ubu1010srv:/var/log# nslookup
> server 192.168.1.37
Default server: 192.168.1.37
Address: 192.168.1.37#53
>
> slashdot.org
Server:  192.168.1.37
Address: 192.168.1.37#53

Non-authoritative answer:
Name: slashdot.org
Address: 216.34.181.45
```

Hey this seems to work without a forwarder. Using a sniffer you can find out what really happens (since the server is not using a cache, not using your 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! I will choose two volunteers to explain this in front of the class.


File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: dns
Expression... Clear Apply
No. Time Source Destination Protocol Info
205 18.355388 192.168.1.37 128.8.10.90 DNS Standard query A slashdot.org
210 18.483161 128.8.10.90 192.168.1.37 DNS Standard query response
211 18.489380 192.168.1.37 199.249.112.1 DNS Standard query A slashdot.org
212 18.522800 199.249.112.1 192.168.1.37 DNS Standard query response
Transaction ID: 0x6826
Flags: 0x8000 (Standard query response, No error)
Questions: 1
Answer RRs: 0
Authority RRs: 9
Additional RRs: 13
Queries
slashdot.org: type A, class IN
Authoritative nameservers
org: type NS, class IN, ns a2.org.afilias-nst.info
org: type NS, class IN, ns b2.org.afilias-nst.org
org: type NS, class IN, ns d0.org.afilias-nst.org
org: type NS, class IN, ns b0.org.afilias-nst.org
org: type NS, class IN, ns a0.org.afilias-nst.info
org: type NS, class IN, ns c0.org.afilias-nst.info
org: type DS, class IN
org: type DS, class IN
org: type RRSIG, class IN
Additional records
0030 00 00 00 09 00 0d 08 73 6c 61 73 68 64 6f 74 03 .....s lashdot.
0040 6f 72 67 00 00 01 00 01 c0 15 00 02 00 01 00 02 org..... ....
0050 a3 00 00 19 02 61 32 03 6f 72 67 0b 61 66 69 6c .....a2. org.afil
0060 69 61 73 2d 6e 73 74 04 69 6e 66 6f 00 c0 15 00 ias-nst. info....
0070 02 00 01 00 02 a3 00 00 15 02 62 32 03 6f 72 67 ....b2.org
0080 0b 61 66 69 6c 69 61 73 2d 6e 73 74 c0 15 c0 15 .afilias -nst....
0090 00 02 00 01 00 02 a3 00 00 05 02 64 30 c0 52 c0 ....d0.R.
00a0 15 00 02 00 01 00 02 a3 00 00 05 02 62 30 c0 52 ....b0.R
00b0 c0 15 00 02 00 01 00 02 a3 00 00 05 02 61 30 c0 ....a0.
00c0 2d c0 15 00 02 00 01 00 02 a3 00 00 05 02 63 30 -....... ......c0

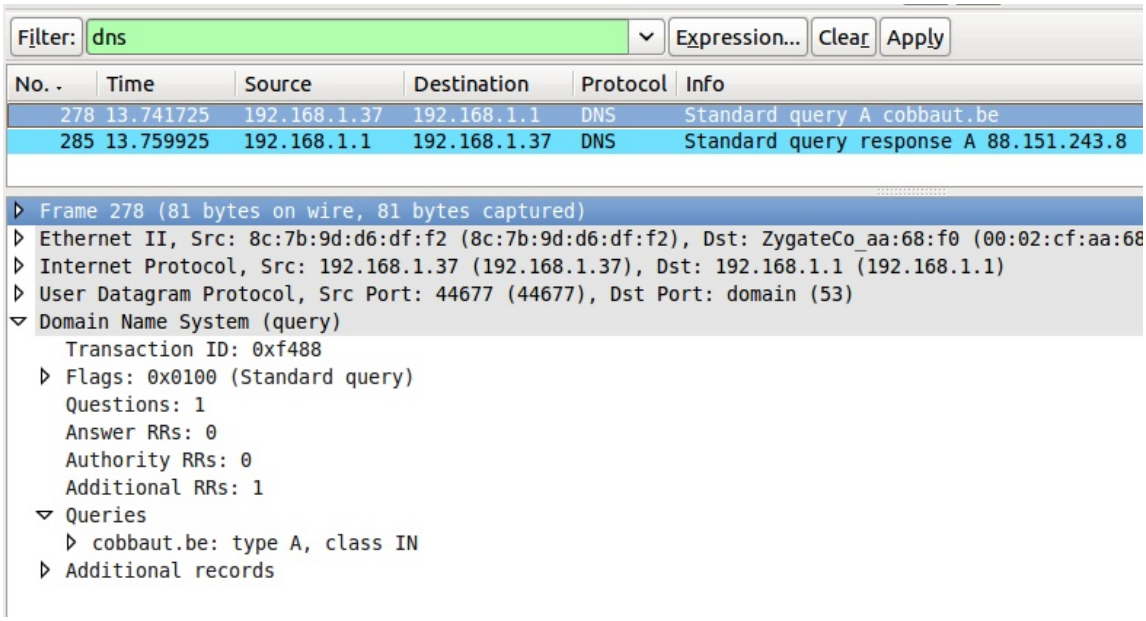
1.13. Practice: caching only with forwarder

5. Add a local dns-server as a forwarder (at my home this is 192.168.1.1, probably different ip in a classroom!).

```
root@ubu1010srv:~# grep -A2 forwarder /etc/bind/named.conf.options| t\
ail -3
forwarders {
  192.168.1.1;
 };
root@ubu1010srv:~# /etc/init.d/bind9 restart
 * Stopping domain name service... bind9                    [ OK ]
 * Starting domain name service... bind9                    [ OK ]
root@ubu1010srv:~#
```

6. Explain the purpose of adding the forwarder. What is our DNS server doing when it receives a query ? Again the wireshark screenshot can help, you should see something similar.

```
root@ubu1010srv:~# nslookup
> server
Default server: 192.168.1.4
Address: 192.168.1.4#53
> server 192.168.1.37
Default server: 192.168.1.37
Address: 192.168.1.37#53
>
> cobbaut.be
Server:  192.168.1.37
Address: 192.168.1.37#53

Non-authoritative answer:
Name: cobbaut.be
Address: 88.151.243.8
```



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 ?

```
root@ubu1010srv:~# cat /etc/resolv.conf
nameserver 127.0.0.1
```

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 ?

1.14. 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 new TLD **.paul** and the domain **cobbaut.paul** and put the information in **/etc/bind/named.conf.local**.

```
root@ubu1010srv:/etc/bind# grep -C1 cobbaut named.conf.local

zone "cobbaut.paul" {
 type master;
 file "/etc/bind/db.cobbaut.paul";
};
```

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 300 seconds).

```
root@ubu1010srv:/etc/bind# cat db.cobbaut.paul
;
; BIND data file for domain cobbaut.paul
;
$TTL 604800
@ IN SOA ns.cobbaut.paul. root.cobbaut.paul. (
                        20110516         ; Serial
                             300         ; Refresh
                             200         ; Retry
                         2419200         ; Expire
                          604800 )       ; Negative Cache TTL
;
@               IN      NS      ns.cobbaut.paul.
ns              IN      A       192.168.1.37
ubu1010srv      IN      A       192.168.1.37
anya            IN      A       192.168.1.1
mac             IN      A       192.168.1.30
root@ubu1010srv:/etc/bind#
```

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

```
root@ubu1010srv:/etc/bind# grep cobbaut /var/log/daemon.log
May 16 00:33:49 ubu1010srv named[25449]: zone cobbaut.paul/IN: loaded\
 serial 20110516
```

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

```
root@ubu1010srv:/etc/bind# ping mac.cobbaut.paul
PING mac.cobbaut.paul (192.168.1.30) 56(84) bytes of data.
64 bytes from 192.168.1.30: icmp_req=1 ttl=64 time=2.28 ms
64 bytes from 192.168.1.30: icmp_req=1 ttl=64 time=2.31 ms (DUP!)
^C
--- mac.cobbaut.paul ping statistics ---
1 packets transmitted, 1 received, +1 duplicates, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.282/2.296/2.310/0.014 ms
root@ubu1010srv:/etc/bind# dig anya.cobbaut.paul

; <<>> DiG 9.7.1-P2 <<>> anya.cobbaut.paul
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38237
```

```
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;anya.cobbaut.paul.  IN A

;; ANSWER SECTION:
anya.cobbaut.paul. 604800 IN A 192.168.1.1

;; AUTHORITY SECTION:
cobbaut.paul.  604800 IN NS ns.cobbaut.paul.

;; ADDITIONAL SECTION:
ns.cobbaut.paul. 604800 IN A 192.168.1.37

;; Query time: 1 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Mon May 16 00:38:22 2011
;; MSG SIZE  rcvd: 84

root@ubu1010srv:/etc/bind#
```

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

```
server os : Ubuntu 10.10
ip : 192.168.1.37
domain name: cobbaut.paul
server name: ns.cobbaut.paul
```

1.15. 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
```

1.16. 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 "cobbaut.paul" {
 type master;
 file "/etc/bind/db.cobbaut.paul";
 allow-transfer { 192.168.1.31; };
};
root@ubu1010srv:/etc/bind#
```

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]#
```

Chapter 2. advanced DNS

Table of Contents

2.1. DNS round robin

When you create multiple A records for the same name, then **bind** will do a round robin of the order in which the records are returned. This allows the use of DNS as a load balancer between hosts, since clients will usually take the first ip-address offered.

This is what it looks like in the **zone configuration file**.

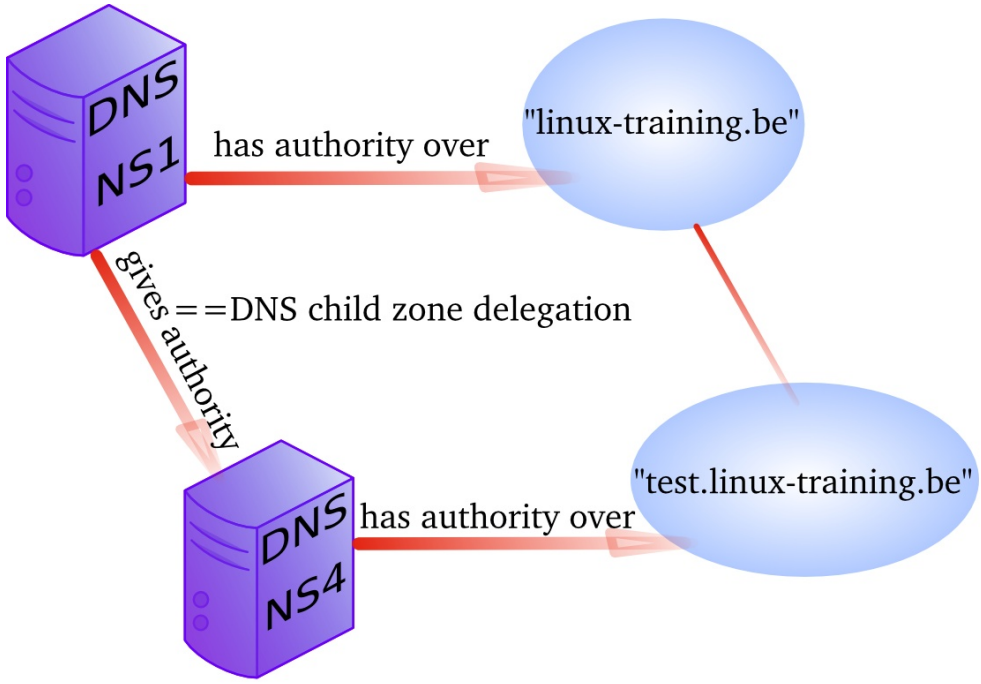
```
faith  IN A 192.168.1.20
faith  IN A 192.168.1.22
```

Below a screenshot of nslookup querying a load balanced A record. Notice the order of ip-addresses returned.

```
> server 192.168.1.35
Default server: 192.168.1.35
Address: 192.168.1.35#53
> faith.cobbaut.paul
Server:  192.168.1.35
Address: 192.168.1.35#53

Name: faith.cobbaut.paul
Address: 192.168.1.20
Name: faith.cobbaut.paul
Address: 192.168.1.22
> faith.cobbaut.paul
Server:  192.168.1.35
Address: 192.168.1.35#53

Name: faith.cobbaut.paul
Address: 192.168.1.22
Name: faith.cobbaut.paul
Address: 192.168.1.20
> faith.cobbaut.paul
Server:  192.168.1.35
Address: 192.168.1.35#53

Name: faith.cobbaut.paul
Address: 192.168.1.20
Name: faith.cobbaut.paul
Address: 192.168.1.22
```

2.2. DNS delegation

You can **delegate** a child domain to another DNS server. The child domain then becomes a new zone, with authority at the new dns server.



This is a screenshot of the zone database file with delegation.

```
root@ubu1010srv:/etc/bind# cat db.linux-training.be
$TTL 3d ; default ttl set to three days
$ORIGIN linux-training.be.
@       IN  SOA  ns1.linux-training.be. paul.linux-training.be. (
 20110524
 300
 300
 10000
 20000
 )
 IN NS ns1.linux-training.be.
 IN NS ns2.linux-training.be.
 IN NS ns3.linux-training.be.
 IN MX 10 smtp.openminds.be.
ns1 IN A 192.168.1.35
ns2 IN A 192.168.1.36
ns3 IN A 192.168.1.37
www IN A 192.168.1.35
mac IN A 192.168.1.30

$ORIGIN office.linux-training.be.
@ IN NS ns4.office.linux-training.be.
; or replace those two lines with:
; office.linux-training.com  IN  NS  ns4.office.linux-training.be

 IN NS ns1.linux-training.be. ; in case this is a slave
ns4 IN A 192.168.1.33  ; the glue record
; ns4.office.linux-training.be A 192.168.1.33 ; also ok!
```

2.3. DNS load balancing

Not as above. When you have more than one DNS server authoritative for a zone, you can spread queries amongst all server. One way to do this is by creating NS records for all servers that participate in the load balancing of external queries.

You could also configure different name servers on internal clients.

2.4. DNS notify

The original design of DNS in rfc 1034 and rfc 1035 implemented a **refresh** time in the **SOA** record to configure a time loop for slaves to query their master server. This can result in a lot of useless pull requests, or in a significant lag between updates.

For this reason **dns notify (rfc 1996)** was designed. The server will now notify slaves whenever there is an update. By default this feature is activated in **bind**.

Notify can be disabled as in this screenshot.

```
zone "1.168.192.in-addr.arpa" {
        type master;
        notify no;
        file "/etc/bind/db.192";
};
```

2.5. testing IXFR and AXFR

Full zone transfers (AXFR) are initiated when you restart the bind server, or when you manually update the zone database file directly. With **nsupdate** you can update a zone database and initiate an incremental zone transfer.

You need DDNS allowed for **nsupdate** to work.

```
root@ubu1010srv:/etc/bind# nsupdate
> server 127.0.0.1
> update add mac14.linux-training.be 86400 A 192.168.1.23
> send
update failed: REFUSED
```

2.6. DDNS integration with DHCP

Some organizations like to have all their client computers in DNS. This can be cumbersome to maintain. Luckily **rfc 2136** describes integration of DHCP servers with a DNS server. Whenever DHCP acknowledges a client ip configuration, it can notify DNS with this clients ip-address and name. This is called **dynamic updates** or DDNS.

2.7. reverse is forward in-addr.arpa

Reverse lookup is actually iomplemented as a forward lookup in the **in-addr.arpa** domain. This domain has 256 child domains (from 0.in-addr.arpa to 255.in-addr.arpa), with each child domain having again 256 child domains. And this twice more to a structure of over four billion (2 to the power 32) domains.

2.8. ipv6

With rfc 3596 came ipv6 extensions for DNS. There is the AAAA record for ipv6 hosts on the network, and there is the **ip6.int** domain for reverse lookup (having 16 child domains from 0.ip6.int to f.ip6.int, each of those having again 16 child domains...and this 16 times.

2.9. split-horizon dns

You can use the **view clause** in **bind** to give different results to different clients.

```
view "antwerp" {
match-clients { 172.16.42/24; }; // the network in Antwerp
zone "cobbaut.paul" {
        type master;
        file "/etc/bind/db.cobbaut.paul.antwerp"; // www=172.16.42.9
        };
};

view "brussels" {
match-clients { 172.16.33/24; }; // the Brussels network
zone "cobbaut.paul" {
        type master;
        file "/etc/bind/db.cobbaut.paul.brussels"; // www=172.16.33.4
  };
};
```

2.10. DNS security : file corruption

To mitigate file corruption on the **zone files** and the **bind configuration** files protect them with Unix permissions and take regular backups.

2.11. DNS security : zone transfers

Limit zone transfers to certain ip addresses instead of to **any**. Nevermind that ip-addresses can be spoofed, still use this.

2.12. DNS security : zone transfers, ip spoofing

You could setup DNSSEC (which is not the easiest to maintain) and with rfc 2845(tsig?) and with rfc 2930(tkey, but this is open to brute force), or you could disable all zone transfers and use a script with ssh to copy them manually.

2.13. DNS security : queries

Allow recursion only from the local network, and iterative queries from outside only when necessary. This can be configured on master and slave servers.

```
view "internal" {
match-clients { 192.168.42/24; };
recursion yes;
...

};

view "external" {
match-clients { any; };
recursion no;
...

};
```

Or allow only queries from the local network.

```
options {
      allow-query { 192.168.42.0/24; localhost; };
};

zone "cobbaut.paul" {
      allow-query { any; };
};
```

Or only allow recursive queries from internal clients.

```
options {
        allow-recursion { 192.168.42.0/24; localhost; };
};
```

2.14. DNS security : chrooted bind

Most Linux distributions allow an easy setup of bind in a **chrooted** environment.

2.15. DNS security : DNSSEC

DNSSEC uses public/private keys to secure communications, this is described in rfc's 4033, 4034 and 4035.

2.16. DNS security : root

Do not run bind as root. Do not run any application daemon as root.

Index

Symbols

A

B

C

D

F

G

H

I

M

N

P

Q

R

S

T

Z