DNS Resolving using *nslookup*

Oliver Hohlfeld & Andre Schröder

January 8, 2007

Abstract

This report belongs to a talk given at the networking course (Institue Eurecom, France) in January 2007. It is based on an experimental setup consisting of a bootable Knoppix Linux DVD which contained a installation of Damn Small Linux (DSL). The DSL machine was connected to Knoopix by a virtual network link. The whole setup aimed to experiment with fundamental networking protocols. The scope of this talk was to experiment with a DNS resolver to carry out name resolutions.

1 Task

Use your favorite command to carry out name resolutions!

2 Possible Ways to Carry out Name Resolutions

There is a bunch of tools available to carry out name resolutions. Furthermore, every application that can handle host names instead of IP addresses to establish connections (e.g. webrowsers) queries the domain name system (DNS). The application simply calls an operating system method (e.g. getHostByName()) to resolve the IP address of a host name in order to be able to establish a TCP connection or send UDP data to the destination host. These name resolutions can be monitored with Ethereal (now WireShark).

However, this mechanism is not suitable if a user wants to get details about a host without using Ethereal. An example for that could be to look for a mail server. For this purpose, tools like *nslookup*, *dig* or *host* were created. Since these tools provide roughly identical functionality, we picked only one of them to present DNS features. We chose *nslookup* which is introduced in the next section.

3 About nslookup

The command-line tool *nslookup* is a DNS resolver. It can be used to query DNS servers, to resolve IP addresses of host names, or to get the corresponding names, mail server, and name servers of a domain. Moreover, *nslookup* is available for Windows and Unix-like operating systems.

It can be used in two modes:

- non-interactive (parameter) mode: in this mode, *nslookup* is controlled by command-line parameters when launching the application. This mode is very useful for scripting.
- interactive mode: no command-line parameters are required in this mode, but the user can control the tool by interactive input. This is useful for some tests since parameters can be set and multiple queries can be executed without re-entering the parameters for each test.

3.1 Selected Commands

This section presents some selected commands to work with *nslookup*:

```
NAME           - print info about the host/domain NAME using default server
NAME1 NAME2    - as above, but use NAME2 as server
set OPTION     - set an option
   [no]debug       - print debugging information
   [no]d2          - print exhaustive debugging information
   [no]recurse     - ask for recursive answer to query
   type=X          - set query type (ex.A, ANY, CNAME, MX, NS, PTR, SOA, SRV)
   class=X         - set query class (ex. IN (Internet), ANY)
server NAME    - set default server to NAME, using current default server
```

Please note that the version of *nslookup* installed on the DSL (Damn Small Linux) machine differs from the version on the Knoppix box in terms of allowed parameters. The version installed on DSL does not support some parameters.

3.2 Usage Examples

In the following sections, some simple tests with *nslookup* are described.

3.2.1 Simple Lookup

The first test will demonstrate a simple DNS lookup by using *nslookup*. The goal is to resolve the host *www.eurecom.fr* to an IP address. The following console output will show the usage of *nslookup*.

```
$ nslookup www.eurecom.fr
Server:  camilla.eurecom.fr
Address:  10.3.2.200

Name:    dayak.eurecom.fr
Address:  10.3.2.240
Aliases:  www.eurecom.fr
```

The host *www.eurecom.fr* could be resolved to *10.3.2.240*.

3.2.2 Looking up different record types

Besides resolving a host to an IP address, other DNS record types can be retrieved. For example, one can get the corresponding mail server for a domain. This record is necessary in order to deliver mail by a mail server.

In the following example, *nslookup* will be used to retrieve the corresponding DNS server responsible for the domain *eurecom.fr*.

```
$ nslookup -type=ns eurecom.fr
Server:  camilla.eurecom.fr
Address:  10.3.2.200

eurecom.fr      nameserver = dns.eurecom.fr
```

The next example demonstrates a query for the *eurecom.fr* mail server.

```
$ nslookup -type=mx eurecom.fr
Server:  camilla.eurecom.fr
Address:  10.3.2.200

eurecom.fr      preference = 0, mail exchanger = smtp.eurecom.fr
eurecom.fr      nameserver = dns.eurecom.fr
```

3.2.3 Detailed Output

nslookup offers two debug options (-debug and -d2) for displaying more detailed output. The output of this options comes close to the kind of information which can be seen in Ethereal (a subset of information is displayed by nslookup).

Produced Network Traffic A traffic analysis with Ethereal shows three queries which cause six packets (query and response):

- Resolve default DNS server name (camilla.eurecom.fr). This is only for displaying reasons and not needed to resolve the host name. This output can be seen in the first two lines of the nslookup response in section *A Simple Lookup*. Most pages call this request a hidden query[1].
- Try local search[2] - *www.leurrecom.org.eurecom.fr*. This fails as the requested host does not exist. Therefore, a third query is issued.
- Try external search - www.leurrecom.org. This query gives the correct answer.

[1]The query is hidden as it's not related to the DNS query the user carried out. It's just an additional query to get the name of the used DNS server. The only reason is to print out its name. Hidden queries are not needed in order to carry out the request of the user. Many pages criticize this fact, as it can be seen as a waste of time. If one really wants to know the name of the used DNS server, one would explicitly carry out this request. There is no reason why it has to be performed on every call of nslookup.

[2]In case a local search domain is specified in the name resolution settings.

***nslookup* Output**

```
$ nslookup -d2 www.leurrecom.org
  [...] (DNS server name request)
  [...] (local request          )
SendRequest(), len 35
  HEADER:
   opcode = QUERY, id = 3, rcode = NOERROR
   header flags:  query, want recursion
   questions = 1,  answers = 0,  authority
   records = 0,  additional = 0

QUESTIONS:
   www.leurrecom.org, type = A, class = IN
```

The first two lines of the output (which are skipped here because this goes too much into details) correspond to the first two queries mentioned in the previous section. First, the DNS server is resolved and after this, a local search for *www.leurrecom.org.eurecom.fr* is issued, which fails. However, after this two lines, the answer to the true query – *what is the IP address of www.leurrecom.org?* – can be seen.

```
Got answer (79 bytes):
    HEADER:
        opcode = QUERY, id = 3, rcode = NOERROR
        header flags:  response, auth. answer,
        want recursion, recursion avail.
        questions = 1,  answers = 1,
        authority records = 1,  additional = 0
    QUESTIONS:
        www.leurrecom.org, type = A, class = IN
    ANSWERS:
    ->  www.leurrecom.org
        type = A, class = IN, dlen = 4
        internet address = 10.3.2.240
        ttl = 172800 (2 days)
    AUTHORITY RECORDS:
    ->  leurrecom.org
        type = NS, class = IN, dlen = 16
        nameserver = dns.eurecom.fr
        ttl = 172800 (2 days)
```

4 Damn Small Linux (DSL) / Knoppix Setup

When booting the virtual DSL machine inside a Knoppix box, all traffic is routed via a virtual link to the outside world: between interface eth0 of DSL and the interface tap0 on the Knoppix machine. The user is able to browse the internet, just as usual. But why is this possible? How does the DSL machine know which name server it has to use or how to route all the packets?

The answer to this question is hidden in the boot procedure of the DSL machine. After the virtual network interface of the virtual DSL machine has been enabled, a DHCP query is sent out to discover a DHCP server. The further network configuration (default gateway to route the packets, or the name server to use) is then handed out by the DHCP server running on the

Knoppix machine. The Knoppix box itself gets the configuration by DHCP as well, e.g. from the Eurecom DHCP server. After that, the DSL machine is able to carry out DNS queries.

The procedure is the following (please note that the DHCP description is simplified since this is out of the scope of this presentation):

- First of all, a virtual link between Knoppix and the DSL machine is established
- The DSL machine does not have an IP address yet. Therefore, a DHCP request is broadcasted to find a DHCP server.
- The DHCP server (running on Knoppix) sends an ARP request to check if a client is already using the IP address it wants to assign to the DSL machine.
- After some more DHCP negotiation, the DSL machine receives an offer from the Knoppix box, containing its new IP address, the default gateway (in order to route packets to the outside world) and the DNS server to be used. (This offer must be acknowledged by the DSL machine)
- The DSL machine is now able to carry out DNS queries.

DNS queries are routed through the Knoppix machine which does a network address translation (NAT) since the local IP address of the DSL machine is only valid in the local area network (LAN) and cannot be used for communication with the outside world.

5 DNS Usage Hints

During our experiments with *nslookup* we discovered some issues which are explained in the following section.

5.1 Norecursive Search & Cache

The first issue concerns non-recursive lookups and the issue of caching query results. A host for that no entry in the Eurecom DNS server existed was selected and a non-recursive query was started. For the selected host *www.doom.de*, *nslookup* presented a list of German name servers which might know the correct answer. No IP address was returned since the queried DNS server was asked not to search recursively which means that he did not ask further DNS servers.

```
$ nslookup -norecursive www.doom.de
Server:  camilla.eurecom.fr
Address:  10.3.2.200

Name:    www.doom.de
Served by:
- Z.NIC.de
          de
- A.NIC.de
          de
...
```

A recursive query finally reveals the IP address:

```
$ nslookup -recursive www.doom.de
Server:  camilla.eurecom.fr
Address:  10.3.2.200

Non-authoritative answer:
Name:    www.doom-labs.net
Address:  83.120.2.4
Aliases:  www.doom.de
```

When querying the same host another time with a non-recursive search, we see that even in this case the correct answer is returned.

```
$ nslookup -norecursive www.doom.de
Server:  camilla.eurecom.fr
Address:  10.3.2.200

Non-authoritative answer:
Name:    www.doom-labs.net
Address:  83.120.2.4
Aliases:  www.doom.de
```

The reason for this effect is that the DNS server that is queried by nslookup stored a temporary entry of the result of recursive search in its cache. As long as this entry exists in the cache, the server can return the correct result immediately without recursively searching.

5.2 Timeout

In some cases, the user may experience timeouts. That does not necessarily mean that no answer can be found. Rather the client didn't wait long enough for an answer. The timeout was set to short in order to retrieve the result. During the recursive query process, several DNS servers might have been queried which takes time.

```
$ nslookup www.doom.dk
Server:  camilla.eurecom.fr
Address:  10.3.2.200

DNS request timed out.
    timeout was 2 seconds.
*** Request to camilla.eurecom.fr timed-out
```

By after a second try, the correct answer is returned!

```
$ nslookup www.doom.dk
Server:  camilla.eurecom.fr
Address:  10.3.2.200

Non-authoritative answer:
Name:    www.doom.dk
Address:  12.162.162.246
```

How come? After the client stopped waiting for an answer, the queried DNS server receives the answer. After the second query, the Eurécom DNS server can immediately return the cached result. A possibility to overcome this problem is to increase the timeout by using a command-line parameter (timeout in seconds):

```
$ nslookup -timeout=6 www.doom.dk
```

6 Virtual Servers / Multiple Names per IP address

When resolving the host names *www.leurrecom.fr* and *www.eurecom.fr* the user will notice that both names are resolved to the same IP address *192.168.106.37*. This is possible because of several A records, which can point to the same IP address. An A record maps a host name to an IPv4 address. The reverse lookup of IP addresses can be done by using PTR records (pointer), which map an IP address to a host name. This host name needs not necessarily to be the same as in the A records! When looking up *192.168.106.37* we get *felicia.eurecom.fr* as result. This is just another name for the Eurecom web server. Furthermore, CNAME entries represent aliases for A records, e.g. *CNAME www.leurrecom.fr www.eurecom.fr*.

Mapping several host names to a single IP address is widely used in the internet, e.g. for so called *virtual servers* (in terms of the Apache webserver). Internet service providers who offer domain packages along with webspace and further services(hosting) often run multiple domains on a single machine. Multiple host names are mapped to the same IP address! But how can the webserver then distinguish between all these hosts, if it always connects to the same IP address? The answer is hidden in the HTTP header where the browser specifies the host it wants to reach.