Computer Networks: Assignment 2

20 February 2016

Solving and submitting you assignment

Requirements about the delivery of this assignment:

- Submit via Blackboard (`http://blackboard.ru.nl`);
- Upload one single zip file containing two files, your solution of the assignment plus the Python traceroute program;
- The solution file as well as the archive file should take the name of your student number, for example student *s0123456* should submit a file named *s0123456.pdf* placed in the archive file *s0123456.zip*. the Python traceroute program should be a separate file in the archive named traceroute.py.
- Write both your name and student number into the document (and only your student number in the filename).

Deadline: Wednesday, March 1, 20:00. sharp!

Goals: After completing these exercises successfully you should be able to:

- know how to use the *dig* network utility;
- get a sense of DNS encoding/compression techniques;
- understand DNS caching and be able to read through a simple DNS database;
- implement your own `traceroute`;
- setup a basic network with NetKit.

Marks: You will be graded with marks from 0 to 3 where 0 means not serious, 1 means serious but insufficient, 2 means sufficient and 3 means good. You can have at most 1 assignment graded 0. To get 1 or more, you MUST attempt to solve ALL exercises, even if the provided solution is not correct/complete. In other words, leaving an exercise out automatically turns your grade to 0. In your solution, please explain all answers clearly and concisely.

1 The dig command for DNS

In this problem, we use the **dig** tool available on POSIX hosts to explore the hierarchy of DNS servers. For Windows users, you will have to download the BIND DNS software, which contains the **dig** utility. You can download the latest version of BIND here `http://www.isc.org/downloads/bind/`.

As depicted in Figure 1, a DNS server higher in the DNS hierarchy delegates a DNS query to a DNS server lower in the hierarchy, by sending back to the DNS client the address of that lower-level DNS server. First read dig's man page[1], and then solve the points. As a remark, the figure was snipped from Microsoft's website, which provides a valuable reference on DNS, see [2].

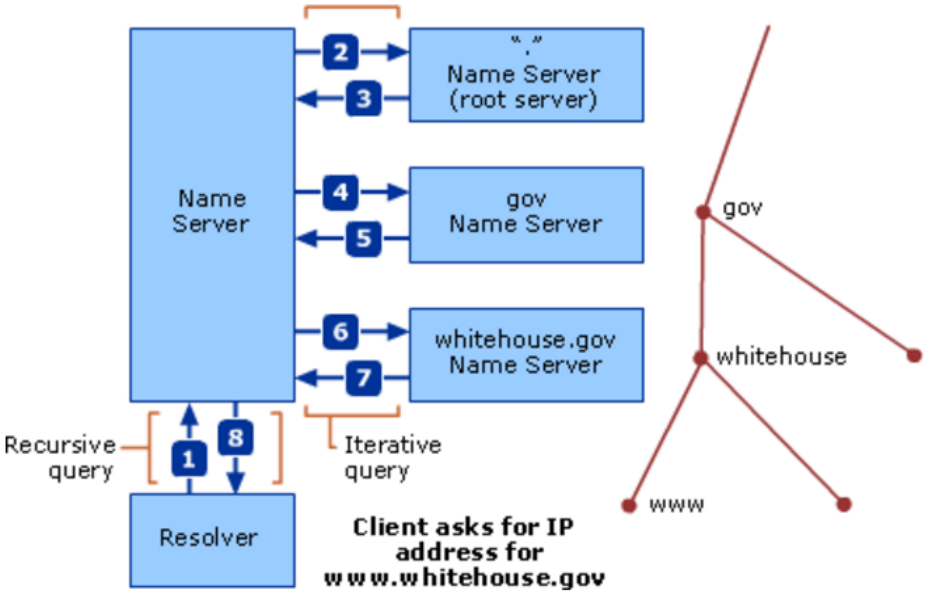

Figure 1: Route from client to server

a) Starting with a root DNS server (from one of the root servers [a-m].root-servers.net), initiate a sequence of queries for *cs.ru.nl*, our department's hostname using dig. For each query, you have to specify the global-server (which is prefixed by @) and the hostname to be resolved. Give a print-out of the output of each query in the chain.

b) For the hostname *cs.ru.nl*, launch a delegation chain using +trace command option. (you no longer have to specify the global-server) Without giving a print-out, explain what happens. Optionally, you can try an online DNS delegation tracer, for example [3].

[1] `ftp://ftp.isc.org/isc/bind/cur/9.9/doc/arm/man.dig.html`
[2] `https://technet.microsoft.com/en-us/library/dd197461(v=ws.10).aspx`
[3] `http://simpledns.com/lookup-dg.aspx`

2 Decoding DNS Messages

DNS messages follow strict message format. Moreover, encoding and compression techniques are applied to make the messages understandable to the receiving end, as well as to reduce the message size. These techniques, as well as the message format, are described in the Chapter 4 of RFC1035[4]. More accessible explanations for encoding/compression can be found online [5].

Figure 2 presents a capture of a DNS Query and its corresponding response. Your task is to decode a DNS exchange and to figure what domain is being resolved and what records are found.

```
00000000  9a ef 01 00 00 01 00 00  00 00 00 00 02 63 73 02   ........ .....cs.
00000010  72 75 02 6e 6c 00 00 ff  00 01                     ru.nl... ..
    00000000  9a ef 81 80 00 01 00 07  00 00 00 00 02 63 73 02   ........ .....cs.
    00000010  72 75 02 6e 6c 00 00 ff  00 01 c0 0c 00 0f 00 01   ru.nl... ........
    00000020  00 01 43 1e 00 10 00 05  03 6d 78 31 07 73 63 69   ..C..... .mx1.sci
    00000030  65 6e 63 65 c0 0f c0 0c  00 0f 00 01 00 01 43 1e   ence.... ......C.
    00000040  00 08 00 05 03 6d 78 32  c0 2c c0 0c 00 0f 00 01   .....mx2 .,......
    00000050  00 01 43 1e 00 08 00 05  03 6d 78 33 c0 2c c0 0c   ..C..... .mx3.,..
    00000060  00 02 00 01 00 01 43 1e  00 06 03 6e 73 31 c0 2c   ......C. ...ns1.,
    00000070  c0 0c 00 02 00 01 00 01  43 1e 00 06 03 6e 73 32   ........ C....ns2
    00000080  c0 2c c0 0c 00 02 00 01  00 01 43 1e 00 06 03 6e   .,...... ..C....n
    00000090  73 33 c0 2c c0 0c 00 01  00 01 00 01 48 31 00 04   s3.,.... ....H1..
    000000A0  83 ae 08 06                                       ....
```

Figure 2: DNS Query/Response capture in hexdump format

3 DNS cache and database

Suppose you access *www.random.cs.nl* via a local DNS server on campus. We assume that the local DNS server knows the address of the root DNS server. When referring to hostnames we refer to the Fully Qualified Domain Name, which comprises the actual hostname (say *www*) and the domain name (say *random.cs.nl*).

a) Provided no caching and iterative resolve of queries, enumerate the hostnames that need to be resolved respecting the order in which they are resolved.

b) Now let's assume the local DNS server implements caching. Of the hostnames to be resolved, choose the one most *and* least likely to be found stored in the local DNS server's cache. Explain your choices.

c) How can you tell via DNS if a site has been recently accessed?

d) Every time a hostname is resolved by a query, the local DNS server caches the response in a DNS record. The record, among others, contains the TTL value which dictates for how long can the cached record be used to resolve hostnames, precluding the need to query a DNS server. Try to relate this TTL value to the likelyhood of a cache hit. (which was discussed in a previous point)

[4] https://tools.ietf.org/html/rfc1035
[5] http://www.tcpipguide.com/free/t_DNSNameNotationandMessageCompressionTechnique.htm

Let's now assume an excerpt of a DNS database for the domain *random.cs.nl.* of the form:

random.cs.nl.	86400	IN	NS	alpha
random.cs.nl.	86400	IN	MX	1. beta
random.cs.nl.	86400	IN	MX	2. delta
alpha	86400	IN	A	130.122.131.10
beta	86400	IN	A	131.122.131.10
delta	86400	IN	A	132.122.131.10
www	86400	IN	CNAME	alpha
ftp	86400	IN	CNAME	alpha
imap	86400	IN	CNAME	alpha
printer	86400	IN	A	134.122.131.10

e) What is the Fully Qualified Domain Name (FQDN) of the printer? How about the IP of the printer?

f) If you send an email to a user within the "random domain", which mail server will handle the mail delivery?

g) We notice multiple CNAME entries, for a www server, an ftp server and an imap server respectively. What would be an advantage of using CNAMEs in this context? (and not A entries)

4 ICMP Traceroute

Along with the assignment, you are given an incomplete ICMP based `traceroute` implementation. It already contains routines necessary to build `ICMP_ECHO_REQUEST` packets with given TTL values and to receive responses back.

You have to fill in the main procedure. Your implementation should take only a destination address as parameter. It then prints the **IP addresses** AND **FQDNs** of the nodes in the path to this address, along with 3 RTT measurements. In case no response was received it should print *. You need not worry about the preciseness of the RTT measurement (for example, you need not start a timer at the exact time of sending). The timeout is set to 1, while the maximum number of packets is set to 10.

The packets you are supposed to handle are of type `ICMP_ECHO_REQUEST`, `ICMP_ECHO_REPLY` and `ICMP_TIME_EXCEEDED`. For information on the structure of each packet, check [6].

Include your implementation in the assignment zip file. Along with the implementation, the assignment pdf should include a brief (yet complete) explanation of how your code works. What are the logical steps and what is what way do they flow? What edge cases exist? This should be centered around the different types of packets sent/received, along with usage of other ICMP fields relevant for the implementation.

[6] `https://tools.ietf.org/html/rfc792`

5 Netkit: your first network

Netkit provides an environment for creating and working with virtual networks. This configuration is done through a terminal in the netkit VM. For example to configure a two host network in netkit you can type

```
vstart A --eth0=domain1
vstart B --eth0=domain1
```

This will open two new command windows (one for each virtual host) and each host will be associated with the network 'domain1'. In these command windows type

```
ifconfig eth0 10.0.0.1 up (for host A)
ifconfig eth0 10.0.0.2 up (for host B)
```

This will set each with an IP address and enable the interface. You will then be able to ping the virtual hosts from each other (ping 10.0.0.1 or 2). You can use the *vlist* command to view all active virtual hosts in netkit.

In this first netkit assignment you are asked to setup three virtual hosts, connect two of them to separate domains and the third to both domains. After starting the virtual hosts you will need to configure them with IP addresses and then have A and B ping C. Please submit a screenshot showing a vlist printout along with the network configuration of each machine.