Brno University of Technology

Faculty of Information Technology

Computer Communications and Networks

2.project (3.variant)

DNS Lookup Tool

April 9, 2018

Stupinský Šimon, (xstupi00)

Contents

Chapter 1

Introduction

This documentation is focused on the description design and implementation of **DNS Lookup Tool.** The main targets of this project were to get acquainted with the DNS systems and problems, which arise at implementation these systems. DNS systems use millions of people around the world daily and not just because of it, is its role in the Internet important.

The primary task of DNS (Domain Name System) is mapping (conversion) of domain addresses (e.g. www.fit.vutbr.cz) to IP addresses (147.229.9.23). Addressing and address translation is an indispensable part of the Internet, without which no communication between users is possible. DNS Lookup Tool is dealing right with translation *IP addresses* and Domain Names.

Chapter 2 deals with the Domain Name System (DNS) architecture. We will get acquainted with the basic building elements of the system, the way of data storage and access to them. We also describe the basic types of records that DNS manages. DNS query resolution and its general processing methods will be introduced in the Chapter 4. Subsequently we will introduced implementation of the DNS Lookup itself in the Chapter 5. We will mention the interesting part of the implementation, introducing the reasons of some solutions of the problems. In the last chapter we will show some examples of how DNS lookup works.

Chapter 2

DNS Architecture

2.1 Overview

The *domain name space*, *DNS servers* and the *resolvers* are three mainly components of which the DNS system is stored. The domain name space contains arrangement and access to data in the DNS systems. DNS servers store this data in their local databases by using the structures, which are part of the Architecture too. Resolver is used to access to data in the DNS. We will now describe the individual components of DNS in more detail [3].

2.2 Domain Name Space

The DNS domain name space is made up of levels. The names of these levels, and the portion of the fully qualified domain name that they represent, are important to know when configuring it. Domain names and IP addresses is quite a lot. To save and search efficiently, the logical space of all domain names is stored in a DNS system in a special way, to the mentioned levels.

- **Top-level domain** ⇒ name of two or three letters used to indicate a country/region or the type of organization using a name, e.g. „`.com`"
- **Second level domain** ⇒ variable-length names registered to an individual or organization for use on the Internet, e.g. „`microsoft.com.`"
- **Sub-domain** ⇒ additional names that an organization can create that are derived from the registered second-level domain name, e.g. „`example.microsoft.com.`"

2.3 DNS Servers

Server DNS is application which manages the data from the Domain Name Space. This space is divided into some zones and relocated into individual DNS servers. Its the main task is answers to queries, which are directed on the DNS database. The data are stored in the form of the sets of *DNS Records*, respectively *Resource Records*.

These records are stored in the local file or are read from the other DNS server with using transform the zone. The information, which the server manages and to which is responsible are called *authoritative*. According to specification are defined two basic types of servers - primary and secondary.

Primary Server DNS, so called *master* or *primary name-server* contains fully records about domains which manages. These records are storing locally in the files. Server provides authoritative (i.e. always exactly) response to these domains. For every domains have exist just one primary name server.

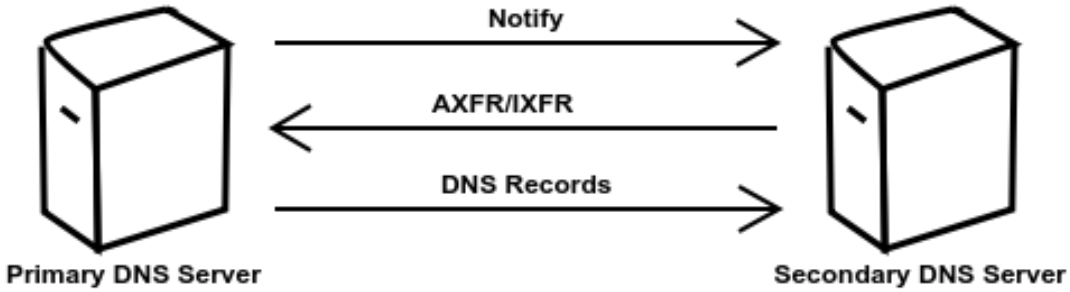

Figure 2.1: The DNS system provides the *Notify3* feature. With this feature a primary DNS provider can notify the secondary providers that the records have changed. After receiving the *Notify* message, secondary servers can use *AXFR4* or *IXFR5* query type to fetch the zone records.

Secondary Server DNS, so called *caching-only nameserver*, obtains data from the primary servers. File, which contains the database of concrete domains (subdomains) is called zone file. The process of transform zone files from the primary server to the secondary is called zone transfer.

2.4 Resolver

The Resolves is the client program, which querying to the data stored in the system DNS. Gradually, we get directly into the area that deals with problematics of this project, which is the acquisition of data from the database DNS. The majority of users program, which need informations from DNS, access this data using by resolver. The main tasks of the Resolver are:

- send queries to server DNS
- interpretation of responses from the servers (error messages, receiving records)
- pass the information to the user program that requested the data.

The Resolver must be able to access at least one DNS server. Answer gets directly from the server or the server returns the link to the next server, where the searched information is saved. Resolver can then forward questions to other servers.

Chapter 3

DNS Protocol

3.1 DNS Messages

All communications inside of the domain protocol are carried in a single format called a message. The top level format of message is divided into 5 sections shown below:

Table 3.1: Format of DNS Messages

Header	**Question**	**Answer**	**Authority**	**Additional**

The header section is always present. The header includes fields that specify which of the remaining sections are present, and also specify whether the message is a query or a response, a standard query or some other opcode, etc.

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
Identification Number															
QR	**OPCode**				**AA**	**TC**	**RD**	**RA**	**Z**			**RCode**			
QDCount															
ANCount															
NSCount															
ARCount															

Table 3.2: Format of Header of the DNS Message

The question section is used to carry the „question“ in most queries, i.e., the parameters that define what is being asked. The section contains QDCOUNT (usually 1) entries, each of the following format:

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	15
0	**QNAME**															
1																
2																
3	**QTYPE**															
4	**QCLASS**															

Table 3.3: Question section format

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format:

	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3	**NAME**
4	**TYPE**
5	**CLASS**
6 7	**TTL**
8	**RDLENGTH**
9 10	**RDATA**

Table 3.4: Resource record format

All concepts of the tables were taken from [1].

3.2 DNS packet compression

The text data compression is occurs at creating the DNS packets. This activity is useful, because some of the chains in the DNS packet repeat such domain name, etc. Therefore, using the pointer to the first occurrence of a string instead of the name. Each domains name is composed from the sequence of chars contains the domain name (i.e. „merlin“, „fit“, „vutbr“, „cz“).

The domain name is stored like as the sequence of pairs (length/value). Each pairs contains length of domains and domains name. This sequence of pairs is ended by zero octet. For example, the domain name „eva.fit.vutbr.cz“ will be written in the packet like as string of chars *(3 eva 3 fit 5 vutbr 2 cz 0)*, where the first byte signed the length of string, which will be following. The sequence is ended by zero char, which is actually the name of the root node in the DNS name tree.

Use of pointers direct to significant reduction of size of the packet. Repeated the occurrences of parts of the domain name in the packet will be changing by pointers on the first occurrences. The array *Offset* determines the distance from the beginning of the DNS packet, i.e. from the first occurrences octet DNS.

Because the first two bytes of length are used to distinguish of string from the pointer, for the length will be using the following six bit. Because the string contains the domain name can be the maximal length equal to $2^6 - 1$, i.e. 63 bytes. The standard also states that the maximum length of the domain name (concatenation of all domains) is 255 bytes.

Chapter 4

Resolution of DNS Query

The resolution is the process of searching the answer in the DNS system. Therefore, that Domain Name Space is structured like as root tree, it suffices to each DNS Server only one information, how to find the arbitrary node in the tree - the address of the root DNS Server. The DNS Server get question to root DNS Server on the highest domain and then follows to tree structure from the root to the node, which contains searching informations.

The root name-server is the authoritative DNS server for the all domains of the highest level TLD. At the query on the arbitrary domain name the root name-server responds by direct searching answer or returns the address of the server DNS, which contains the needed informations. The DNS servers for domains of the first level contains informations about relevant sub-domains of the second level, etc.

The root name-server is irreplaceable for the proper running of the entire DNS system. For this reason, there are thirteen root servers with [a-m].root-servers.net., which are located around the world at different operators. Typically, this is not a single computer. Each server root load is spread over multiple machines. Yet each one answers thousands of queries every second.

4.1 Recursive Query

The **recursive query** is the similar to recursion, like as we known its from the programming. The resolver send query to a specific data in the DNS tree to specific DNS server. The DNS server have to answered on the query the request data or send error message, for example in the case, when it does not known answer. In the case, when the server is not the authoritative server to searching data, have to question other servers and find the authoritative answer. Server can send query to some from the authoritative server and waiting to the answer. In the other hand can send iterative query and obtains link to the other server, which have the answer. Whether the server supports recursive or iterative querying depends on its configuration.

The DNS server, which has received query, resend always the same query on other servers. Never ask for NS records for the domain you are looking for. Most programs (for example, nslookup, dig) sends recursive queries.

4.2 Iterative Query

The **iterative query** saving the works on the side of the DNS server. The server at this kind of querying return the best answer, which it can return. He does not ask more. The interrogated DNS server looks into its local database. If it does not find the answer, it returns the addresses of the servers that are the closest to the address you are looking for.

4.3 Resource Records

DNS records (resource records, RR) are used to store information in the DNS data space. The records are stored in text format in zone files on DNS servers. The most common are Type **A** records that map the domain name to the IP address (so called direct mapping) and the **PTRs** for the reverse (reverse) mapping. Besides these two records are still on the look others, that will be used in the DNS lookup implemented by us and concrete it will be records of the type **AAAA**, **CNAME** and **NS** [5].

- **A** ⇒ include direct domain mapping to IP addresses. It is the only record that contains the IP address on the right side
- **AAAA** ⇒ maps domain addresses to IP addresses of version 6 (IPv6). Writing is quite simple, just like with records A.
- **PTR** ⇒ performs reverse mapping. This means that it converts a numeric IP address to a domain name. Reverse addresses are stored in a special domain in-addr.arpa. in DNS database
- **CNAME** ⇒ assigns to the alias the canonical (official) name of the computer. Each time the DNS server finds a CNAME record in the search, it replaces the canonical domain name name and continues searching. The alias defined by the CNAME record must never stand on the right side of the record. All other records must be mapped to the official (canonical) name, not the alias.
- **NS** ⇒ determines the authoritative server for the given domain name. Using these records, hierarchical DNS structure is building. This record that is located in the given DNS tree node contains pointers to the node followers

Chapter 5

Implementation

In this chapter will be introduce implementation above mentioned **DNS Lookup Tool**. The theoretical foundations was laid in the previous chapters. Our task was created Lookup tool, which will be translate *domain name* and *IP addresses* according to the request of users. Tool have support both types of queries, that is *iterative* and *recursive*. It will be able to handle the requirements of the 5 mentioned records, i.e. *A, AAAA, CNAME, NS* and *PTR*. Tool has maximal *TIMEOUT*, which signed the interval, how long will be tool waiting to the response from the DNS server.

In the initial setup tool executing the recursive query, the value of timeout is set on five seconds and the records type is set on value 1, i.e. record type A. We look at the most interesting parts of the implementation, which contains two main modules. The first of them is model *ipk-lookup*, which representing the main logic of the tool. The second auxiliary module is *stack* and representing the data structure, to use when communicating with DNS servers.

5.1 Design of application

The main idea of this tool is communication with DNS servers, respectively sending queries to the server side. For providing the communication with DNS server was used *BSD sockets*. This sockets are used to communicate with the servers through *UDP protocol*. Based on the selected UDP protocol, port with number **53** was selected. Communication with the server was augmented by connection by **IPv6** address. The address of DNS server can be in *IPv4* form or *IPv6*.

The communication itself is represented by the *queries* from the tool side and *response* from the server side. For its representation was designed data structure, which representing individual parts of DNS Message, as described in the Chapter 2.3. Our design contains four data structure, which gradually represented *DNS Header*, *Question format*, *Response format* and abstraction of the *Resources Records*. The concept of this design was inspiration from [4].

DNS Header is filled by the our tools, before the send to the server side. Enable settings that depend on the current request such as **Recursion Desired**, which depend on the type of executed queries. The another one is flag for identification of query, which have to unique. Of course, do not forget to fill the structure, which representing itself *query*. The query class is set to value one, which representing of **Internet** class. The second, most important

item to sets, it is query type. It responds to the current request and is set to the value of relevant records.

5.2 Iterative queries

The execution of iterative queries was inspiration by tool *DIG* [2]. This tool requests to the DNS server and he will give him a set of *authoritative servers*, which are closer to the answer. The next step is obtaining the address of this authoritative servers and subsequently querying to the one from these servers. These steps are repeated, until it get the request response or it ends in failure. Our way is promoted to the higher level.

The first query is directed to given server by user. The query type of this message is **NS** and the requested domain name is **.**, which means obtaining the *Root DNS servers*. Together with these servers in *Answer Records*, will be returns in the *Additional Records* the IPv4 and IPv6 addresses of some of these servers. Subsequently will be searching the match between obtained address and authoritative servers for additional queries. After the searched this pair of matching, will be send the query to this server. The query type will be set according to users choice. This process will be repeated until the timeout is exhausted, receipt of the required response or exhaustion of all possible servers.

Against to dig tool, our tool will first try to get address of the some og authoritative servers from Additional records and only in the case, that none of the servers will have an address there will send the query with request to address of some server. In practice, this means reducing queries by more than half compared with this tool. Our chance of discovering addresses in the Additional records increased almost double with adding the IPv6 communication too.

5.3 Stack data structure

We have implemented the stack with the data structure of a one-way list. The stack structure of the moves the pointer to the item that is currently at the top. A stack entry is a pointer to a data type void and pointer 'to the next element. The fact that we have the stack item decided to implement using the pointer to the data type void was influenced by the fact that this option allows a wide use of the cartridge. We use the tray properly in several cases.

The first use is for storing the authoritative servers, which was obtained from the Authority Records. These servers will gradually select from the top of the stack and its address are searching. This way we also get to the second use of this data structure. Into the stack are pushed the item obtained from the Additional Records, respectively pairs of type address and relevant name-server of this address.

5.4 Interesting Facts

The solution of the expiration timeout is noteworthy. Failure in our case does not mean, that the desired result will not be achieved. The process is repeated a total of three times and only after triple of failure is unsuccessful reported to users. This decision was made to increase the chances and achieve success at querying to DNS server. Our goal to executing the request of the user has always been to achieve the success at any cost.

Chapter 6

Demonstration of the activities of the applications

A record

```
$ ./ipk-lookup -s 8.8.8.8 www.fit.vutbr.cz ; echo $?
www.fit.vutbr.cz IN A 147.229.9.23
0
```

AAAA record (iterative)

```
$ ./ipk-lookup -s 127.0.1.1 -t AAAA -i www.fit.vutbr.cz. ; echo $?
. IN NS e.root-servers.net
e.root-servers.net IN AAAA 2001:500:a8::e
cz IN NS d.ns.nic.cz
d.ns.nic.cz IN A 193.29.206.1
vutbr.cz IN NS pipit.cis.vutbr.cz
pipit.cis.vutbr.cz IN AAAA 2a01:430:120::4d5d:db6e
fit.vutbr.cz IN NS kazi.fit.vutbr.cz
kazi.fit.vutbr.cz IN A 147.229.8.12
www.fit.vutbr.cz IN AAAA 2001:67c:1220:809::93e5:917
0
```

A record

```
$ ./ipk-lookup -s 8.8.8.8 www4.fit.vutbr.cz. ; echo $?
www4.fit.vutbr.cz IN CNAME tereza.fit.vutbr.cz
tereza.fit.vutbr.cz IN A 147.229.9.22
0
```

CNAME record

```
$ ./ipk-lookup -s 8.8.8.8 -t CNAME www4.fit.vutbr.cz. ; echo $?
www4.fit.vutbr.cz IN CNAME tereza.fit.vutbr.cz
0
```

AAAA record (bad QTYPE)

```
$ ./ipk-lookup -s 8.8.8.8 -t AAAA www4.fit.vutbr.cz. ; echo $?
www4.fit.vutbr.cz IN CNAME tereza.fit.vutbr.cz
1
```

PTR record

```
$ ./ipk-lookup -s 8.8.8.8 -t PTR 2001:67c:1220:8b0::93e5:b013 ; echo $?
3.1.0.b.5.e.3.9.0.0.0.0.0.0.0.0.b.8.0.0.2.2.1.c.7.6.0.1.0.0.2.ip6.arpa
IN PTR merlin6.fit.vutbr.cz
0
```

PTR record (iterative)

```
$ ./ipk-lookup -s 127.0.1.1 -t PTR -i 147.229.13.238 ; echo $?
. IN NS e.root-servers.net
e.root-servers.net IN AAAA 2001:500:a8::e
in-addr.arpa IN NS f.in-addr-servers.arpa
f.in-addr-servers.arpa IN A 193.0.9.1
u.arin.net. IN A 204.61.216.50
ns.ces.net. IN A 195.113.144.233
13.229.147.in-addr.arpa IN NS rhino.cis.vutbr.cz
rhino.cis.vutbr.cz IN AAAA 2001:67c:1220:e000::93e5:30a
238.13.229.147.in-addr.arpa IN PTR pckucerajan.fit.vutbr.cz
0
```

PTR record (iterative) - IPv6

```
$ ./ipk-lookup -s 2001:4860:4860::8888 -t PTR -i 147.229.13.238 ; echo $?
 IN A 198.41.0.4
e.in-addr-servers.arpa. IN A 203.119.86.101
u.arin.net. IN A 204.61.216.50
ns.ces.net. IN A 195.113.144.233
rhino.cis.vutbr.cz. IN A 147.229.3.10
238.13.229.147.in-addr.arpa IN PTR pckucerajan.fit.vutbr.cz
0
```

IPv6 example

```
$ ./ipk-lookup -s 2001:4860:4860::8888 www.fit.vutbr.cz ; echo $?
www.fit.vutbr.cz IN A 147.229.9.23
0
```

Invalid IPv4 address

```
$ ./ipk-lookup -s 127.0:25 www.fit.vutbr.cz ; echo $?
ERROR: Invalid IPv4 address.
1
```

Domain Error

```
$ ./ipk-lookup -s 127.0.1.1 -t PTR -i 147.229.p ; echo $?
. IN NS e.root-servers.net
e.root-servers.net IN AAAA 2001:500:a8::e
in-addr.arpa IN NS f.in-addr-servers.arpa
f.in-addr-servers.arpa IN A 193.0.9.1
y.arin.net. IN A 192.82.134.30
ns.ces.net. IN A 195.113.144.233
ERROR: DNS Server: Non-Existent Domain
1
```

PTR record (on the second chance)

```
$ ./ipk-lookup -s 127.0.1.1 -t PTR -i 147.229.13.238 ; echo $?
. IN NS g.root-servers.net
g.root-servers.net IN A 192.112.36.4
in-addr.arpa IN NS f.in-addr-servers.arpa
f.in-addr-servers.arpa IN AAAA ::

ERROR: Some error was occurred at receiving to Socket!
I'll try it once (2 chances)!

. IN NS a.root-servers.net
a.root-servers.net IN AAAA 2001:503:ba3e::2:30
in-addr.arpa IN NS e.in-addr-servers.arpa
e.in-addr-servers.arpa IN A 203.119.86.101
x.arin.net. IN A 199.71.0.63
ns.ripe.net. IN A 193.0.9.6
gate.feec.vutbr.cz. IN A 147.229.71.10
238.13.229.147.in-addr.arpa IN PTR pckucerajan.fit.vutbr.cz
0
```

IPv6 Error

```
$ ./ipk-lookup -s 2001:4860:4860:8888 -t PTR -i 147.229.13.238 ; echo $?
IPv6 not in presentation format.
1
```

Users Help Message

```
$ ./ipk-lookup -s 127.0.1.1 -t PTR -i ./ipk-lookup -h
IPK-LOOKUP.
Usage: ./ipk-lookup [-h]
       ./ipk-lookup -s server [-T timeout] [-t type] [-i] name

MANDATORY OPTIONS:
    -s server -> DNS Server (IPv4 or IPv6 address) to which the query will be sent.
       name -> The translated domain name, for the -t PTR parameter,
                                the incoming program expects the IPv4 or IPv6 address.
OPTIONAL OPTIONS:
    -T TIMEOUT -> Timeout (in seconds) for query, default value = 5s
    -t type -> Type of query: A (default), AAAA, NS, PTR, CNAME.
    -i iterative -> Enforcing of iterative resolution.

Format of the output: <name> [ ]+ IN [ ]+<type> [ ]+<answer>

EXAMPLES OF USAGE: $ ./ipk-lookup -s 8.8.8.8 -t AAAA -i www.fit.vutbr.cz.
                   $ ./ipk-lookup -s 8.8.8.8 -t PTR -i 147.229.13.238
```

Run-time error in Server

```
$ ./ipk-lookup -s 127.5.1.1 -t PTR -i 147.229.13.238 ; echo $?

ERROR: Some error was occurred at receiving to Socket!
I'll try it once (2 chances)!


ERROR: Some error was occurred at receiving to Socket!
I'll try it once (1 chances)!


ERROR: Some error was occurred at receiving to Socket!

ERROR: Obtaining information about 147.229.13.238 was not succesfull!
1
```

Bibliography

[1] Domain names - implementation and specification. RFC 1035. November 1987. doi:10.17487/RFC1035.
Retrieved from: https://rfc-editor.org/rfc/rfc1035.txt

[2] Consortium, I. S.: Manual Reference Pages - DIG (1). online.
Retrieved from: https://www.gsp.com/cgi-bin/man.cgi?section=1&topic=dig

[3] Matoušek, P.: Systém DNS. online. April 2018. course VUT FIT ISA.

[4] Moon, S.: DNS Query Code in C with linux sockets. online.
Retrieved from:
https://www.binarytides.com/dns-query-code-in-c-with-linux-sockets/

[5] Olafur Gudmundsson, R. B.: Domain Name System (DNS) Parameters. online.
Retrieved from:
https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml