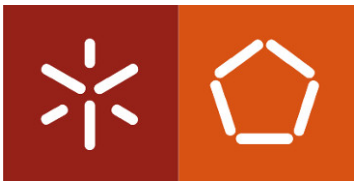

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Curricular Unit of Comunicação por Computador

Academic Year of 2022/2023

Relatório TP2a

Grupo 4

A81384 André Coelho Mota
A96326 Bernard Ambrósio Georges
A96561 Rodrigo Manuel Matos Pereira

November 23, 2022

Contents

List of Figures

1 Introduction

In the context of the *Comunicação por Computador* (CC) subject, we were assigned the task of implementing a DNS service with its client. The present document intends to, throughout its chapters, show how we attacked the assignment and the decisions taken to solve its challenges. Firstly, we will collect the functional requirements of the project and unveil them. Then we will gather all the information to correctly and more safely come to an implementation.

1.1 *What is a DNS?*

A domain name system (DNS), as of the RFC 1034, is a protocol that intends to facilitate to people using the internet access to it. If DNS didn't exist, everytime anyone accessed a website, they would have to know the IP address of a server existing within its domain. DNS facilitates internet access for the common user, i.e., the not *tech savy*, "I just want to be able to use it" kind of user.

Previous domain naming conventions have been used before, such as having a text file (HOSTS.TXT), but having to download these files ended up adding too much overhead to the demands the dawn of the internet required.

These reasons led to the creation of the naming convention we know today, the DNS, which is the main *name solving* system used today.

1.2 A brief introduction to communication protocols

Throughout this report, some other internet protocols will be referred, such as user datagram protocol (UDP) and transmission control protocol (TCP), since they will be utilized to provide communication between different parts of our system.

1.2.1 UDP

UDP (RFC 768) is an internet communication protocol known to be simpler and create much less communication overhead. Unlike TCP, which will be explained, this protocol leaves all verification to the side of the application that uses it, i.e., it's the application's duty to make sure packets have arrived.

1.2.2 TCP

TCP (RFC 793) is an internet communication protocol known to be very reliable and to have ways to ensure that its packets have arrived. It's reliability is much better than UDP's, and because of this, it will also create more communication overhead, since one of the ways it ensures a communication is successful consists on acknowledging it 3 times.

2 System's Architecture

The base of our architecture will be a topology that will have several servers and several clients communicating between each other.

The clients will communicate with the servers, requiring them to resolve a domain's name, and if such is stored in the main database, the request will be answered and fulfilled back to the client.

2.1 Topology

At this phase the group decided to start of with an abstract drawing of the topology, representing only the network domains and its connection with the top server and the reverse server.

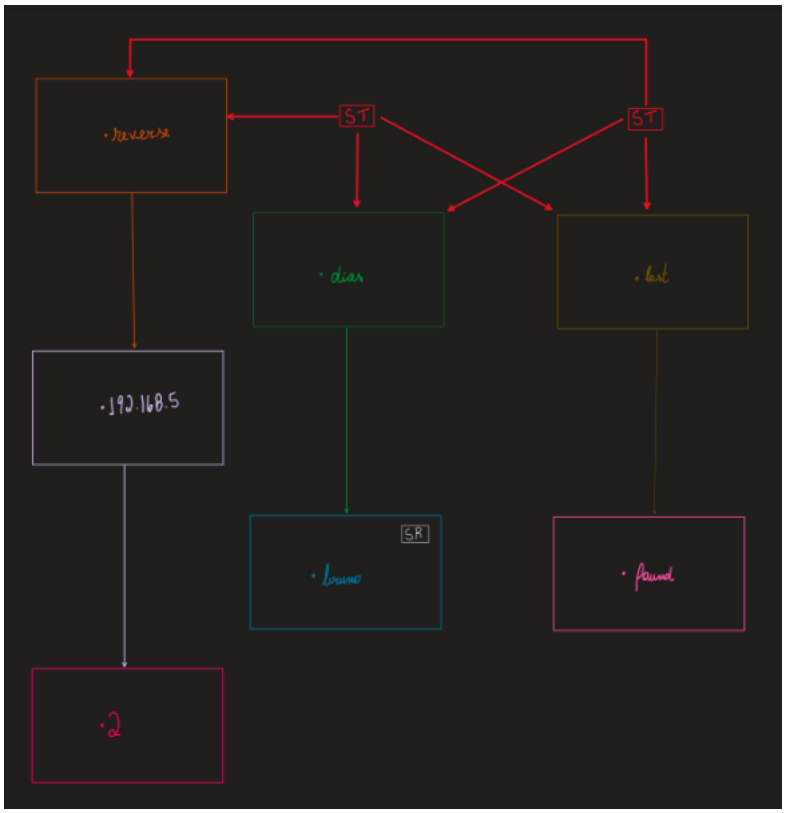

Figure 2.1: Abstract drawing of the topology

In the representation above we were able to see the number of top domain servers the organization of the network in a simpler way.With this in mind we transitioned to core with the version we implemented for the first review. Even though the network above is functional for the long run we desired

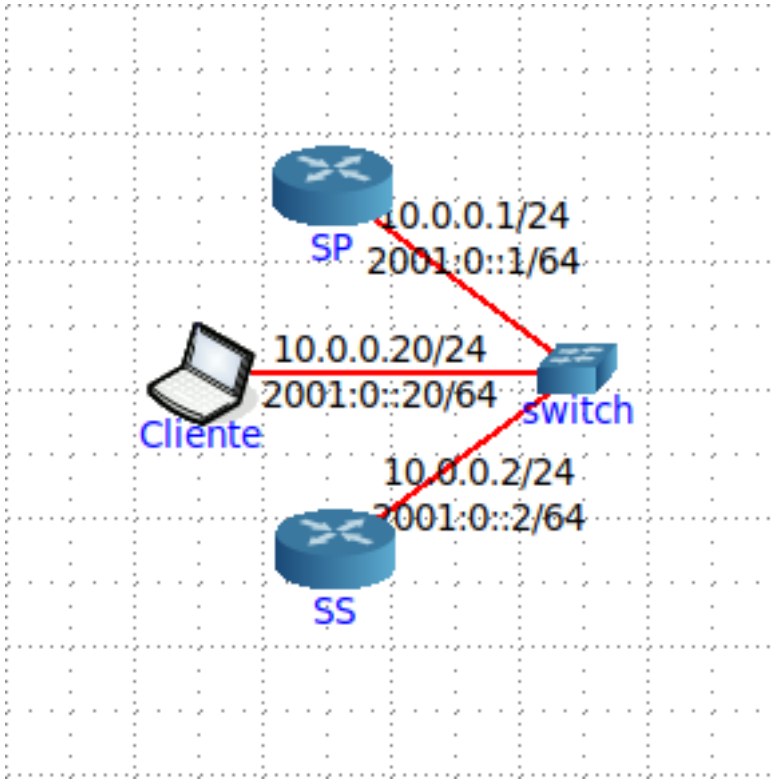

Figure 2.2: Intermediate topology

to create the complete version, represented in the first sketch. In this scope, we attributed for each domain: one primary server (SP) and two secondary server (SS). Beyond the domains we implemented two top servers and a response server (SR).

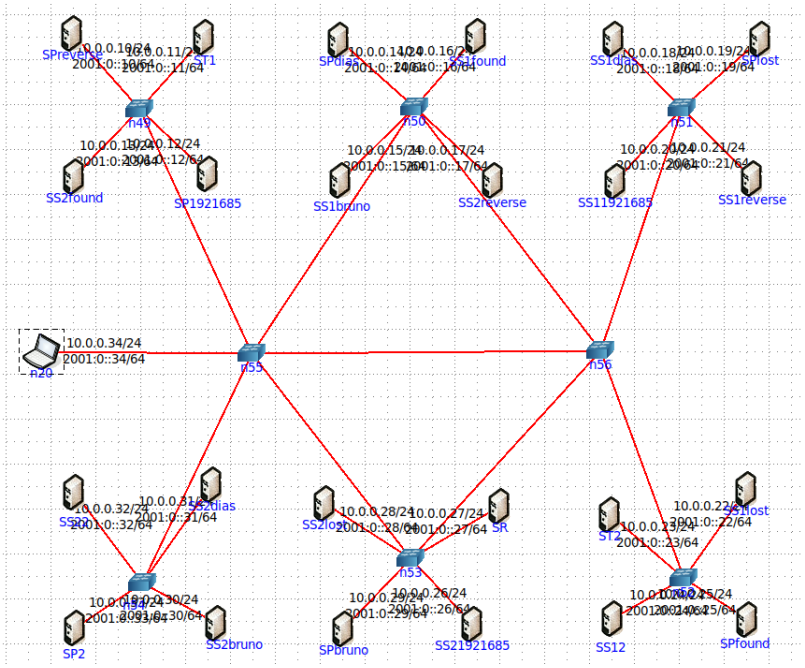

Figure 2.3: Complete network topology

2.2 Components

As it was previously mentioned, the system will be comprised of several different servers. Those can be divided into 3 different categories: Primary Server (SP), Secondary Server (SS) and Response Server (SR). This, in a first implementation, will be 2 different software pieces, as the SP will be the only one with a special implementation. This can, however, change.

The clients will also be implemented, although these will be several instances of the same unique software piece.

2.2.1 Primary Server (SP)

The SP is the heart of the system, and as such, it will have at least in a first instance, its own unique implementation. As previously mentioned the reasons why that is are the fact that it will be the only type of server with direct access to the database, and because of it, the only one with permanent storage.

Because of this uniqueness, of the SP, the SS will need a direct connection. In turn, this must have a connection TCP for this specific connection. As for the rest it will be done by UDP. Its configuration will be done solely at the startup of the system, and it will be via a specialized configuration file with the following format:

2.2.2 Secondary Server (SS)

The SS is a redundancy server, meaning its purpose is to attend requests while the SP is too busy, or to avoid being too busy altogether. It has either a full or partial copy of the SP's database, however,

```
# Configuration file for primary server for example.com
example.com DB /var/dns/example-com.db
example.com SS 193.137.100.250
example.com SS 193.136.130.251:5353
example.com DD 127.0.0.1
example.com LG /var/dns/example-com.log
all LG /var/dns/all.log
root ST /var/dns/rootservers.db
```

Figure 2.4: Configuration File of an SP server

storing it in volatile storage, loosing any stored domains on shutdown.

Similarly to the primary, the configuration is done only on startup and via a specialized configuration file, also this must implement both connections existent on the SP and with the reasons being equally valid. The implementation of this server will be the same as the SR, meaning the only difference will be patented on its configuration, shown bellow:

2.2.3 Response Server (SR)

The SR will receive all requests from clients. As it maintains a smaller copy of the domains existing at the SP, if the domain exists in cache, it will respond directly. In parallel to the SS its cache is a volatile storage.

Since its implementation will be the closelly linket to the SS, it will need TCP sockets to communicate with SS and UDP sockets to communicate with clients, the only difference will be in the fields of the configuration file, as shown bellow:

2.2.4 Configuration Files

The configuration files of the servers above are all very similar to the following format: The configuration of each server will be done on startup, and it will be made by reading a configuration file. The configuration lines will be applied to the field its first argument refers to, being possible fields, this fields dictate its types:

- **Domain name:** indicates the following line arguments will be applied to the entire domain.
- **all:** the following arguments will be applied to all server components not related to domains.
- **root:** the following arguments should be applied to the server's root.

Following these, all configuration files will also have similar parameters complementing these lines:

- **DB:** indicates the path of a database file. This parameter necessarily denotes a primary server.
- **PS:** indicates the IP[:port] address of the primary server of the domain indicated on the parameter (first argument).
- **SS:** indicates the IP[:port] address of a secondary server of the domain indicated on the parameter. This makes this server assume the role of primary server for this domain, and it has authorization to request transmission of information from the database (zone transfer). Multiple entries for the same parameter can exist (one for each domain's SS).

- **DD:** indicates the IP[:port] address of an RS, SS or SP of the domain indicated by parameter. RS using this parameter indicate which domains the servers indicated should contact directly if they receive queries for these domains (when the response isn't in cache), instead of contacting TS. Various entries with the same parameter can exist, one for each default domain server. PS or SS using this parameter indicate which are the only domains they answer to, whether the response is in cache or not, meaning PS or SS only respond to queries indicated by domains in this parameter.

- **TS:** indicates the file listing the TS (the first argument should be *root*).

- **LG:** indicates which log file the server should use to registate activity related to the domain on the first argument. Only domains to which the server is PS or SS should be indicated. All servers should also have an entry to activity unrelated to the domains specified in other LG entries (having the parameter *all*).

2.2.5 Client (CL)

The CL is the program that will send queries to the entire system. It has no storage of any kind, it only provides a terminal user interface (TUI) for the user to interact with and send requests through the system.

Its implementation will need UDP sockets only, as it doesn't require any internal transfer.

2.3 Development notes

2.3.1 Decisions about servers

Considering we have 3 different servers which the differences are storage and the order in which they communicate, a deliberation within development took place. Such effort ended up deciding we would only need 2 different programs, at most. Despite deciding that changes could be made in a future implementation at the same time, we ended up, later in development, deciding doing all 3 servers with the same executable, since they use the same configuration file format.

3 Information Model

One of the crucial aspects in this system is how it will process DNS queries. This dictates how well the CL works, as it will influence how efficiently requests can be processed, and how fast they arrive to its client.

This isn't, however, the only type of information that will be intervening, as internal information vital to how the system is going to be configured will also be imperative to how the system works. For these, we have different kinds of configuration files which we will be describe during this chapter.

3.1 DNS querying

The DNS queries will be provided by the client within its user interface prompt, and this is the only way queries will be inquired to the system. Each query will have a format and a type of which it can be, so the servers can recognise the clients requests.

3.1.1 Query format

A query format changes with the type os the value name but all have priority queuing: Its good to note as well that the SOA formats are meant only for the SS and that the types that are represented in red were not implement at early stage.

- **MX:** The query is made with and e-mail server name.
- **A:**The query is made passing and IPv4 address of a host/server.
- **NS:** The query is made passing a primary or secondary server.
- **PTR:** The query is made with the name of an SP or SS.
- **SOASERIAL:** The query is made with the serial number of the data base of a SP.
- **SOASP:** The query gives the full name of the SP server
- **SOAADMIN:** The query gives the full email of the domain administrator
- **SOAREFRESH:** This query gives the SS server the time he has to as the SP its data base serial number.
- **SOARETRY:** complementaraly to the one above in this case it give the time to retry to receive the query asked.
- **SOAEXPIRE:** This indicates the time the SS has to consider his data base as valid.

- **CNAME:** Not yet implemented
- **DEFAULt:** Not yet implemented

3.2 Server's cache

Notes before reading: this entire section was thought and had our careful deliberation. However, as other parts were of more important implementation during this first deadline, none of the current section was yet implemented.

When a query is sent by a client, it will be processed by the servers that receive it. To make this kind of request easier, this system will have a cache to save previously done requests, achieving faster response times by doing so.

The cache could became too large, to prevent this, it is implemented a timer for the entry to stay valid, to do so it is stored the time of which the request is made and everytime a new entry is made it verifies older ones substituting invalid ones.

3.2.1 Positive cache

This is a mandatory caching module for every DNS system, it's the best way to achieve faster response times. This system will save all previously done requests as well as it's response. This is done so it can provide an answer immediately without having to travel upwards in the system's topology hierarchy. Reducing the delay time caused by request indirection to requests that received a positive answer.

3.2.2 Negative cache

Very similarly to the previously mentioned caching system, this module will also store requests, only instead, if files the ones that could not be properly fulfilled. This will be a separate memory space from the *positive cache*.This is done with the response time being the top priority. With this it is planned that both positive and negative requests won't be negatively impacting each one's response time.

3.3 Known domains

3.3.1 Domain's storage

So every server doesn't need to check whether it knows a domain or not on every access, a structure specialized in storing previous domains will be conceived for every server. This structure will consist mainly of a key/value pair structure (in our case a dictionary). The key will be the domain we're searching and the value will be the parent domain of its key. The search will be iterative within this structure, i.e., it will search while it can find a key/value pair for the subsequent values. The search's stop will be met when the domain in question isn't known, to which a request asking whether the domain is known will be sent to the last known saved domain.

3.3.2 Primary server's case

Exceptionally, this server will be able to store its known domains in permanent storage. The memory module will store in related fashion as other servers, with the difference being the structure. The known domain's will be serialized and loaded on startup for every SP existing in our system.

3.3.3 SP configuration

The primary server will have the biggest configuration file out of all of the server kinds, as it will be the only one specifying a database. This database file should also be specified on the first line, so the server module can distinguish whether it's starting a primary server or another server kind.

With this, one can specify one configuration file for the primary server of **thisdomain.dom**:

```
# this is a primary server's configuration file
thisdomain.dom DB /somepath/dbfile.db
thisdomain.dom SS 192.168.1.20
thisdomain.dom SS 192.168.1.22:73
thisdomain.dom DD 127.0.0.1
thisdomain.com LG /somepath/locallogfile.log
all LG /somepath/mainlog.log
root ST /somepath/rootservers.db
```

3.3.4 SS configuration

The secondary server's configuration will have some elements in common with the primary server, except those concerning databases and stored domains. These will, instead, be dinamically added during connections since memory on this kind of server exists only during execution, as previously mentioned.

Having mentioned this, a configuration file for this server, sharing the same domain **thisdomain.com** can be defined as follows:

```
# this is a secondary server's configuration file
thisdomain.dom DD 127.0.0.1
thisdomain.com LG /somepath/hostlogfile.log
all LG /somepath/programlog.log
```

3.3.5 SR configuration

```
# this is a response server's configuration file
thisdomain.com LG /somepath/hostlogfile.log
all LG /somepath/programlog.log
```

3.4 *log* files

The *log* file will keep tabs on every server, query, client, and many more. Its format will be the same throughout the entire system. The client won't have logging.

3.4.1 *log* fields

- **QR/QE**: a query was sent/received to/from the presented address. The relevant data must be the included in the query. The same entry data used in the query PDU is the same used in the debug mode of communication between elements.
- **RP/RR**: It was sent/received and answer to a query to the indicated adress. the entry values must be the data relevant to the answer.
- **ZT**: indicates a zone transfer with the address of the receiver as the value must be known and the log must have its type as an argument.
- **EV**: It occured an event with the address being shown or the value 127.0.0.1 with the data including adicional information about the event.
- **ER** Occurred and error in the decoding of a data unit.
- **EZ** There was an error in a successful zone transfer.The adress must be of the receiving server and there must be it's role.
- **FL** There was an functioning error the data must include adicional information about its situation.
- **TO** Timeout was detected in the server that was addressed with information specifing the type of such
- **SP** A component was stop the address must indicate 127.0.0.1 and the aditional information being the reason it stoped.
- **ST** A component was started with the address indicating a value of 127.0.0.1 and ther must be the port that is using, the timeout being used e the mode that its using

4 Communication Model

4.1 DNS Header

To transfer the data in an efficient way, either space and complexity wise, we decided to implement a header of this format:

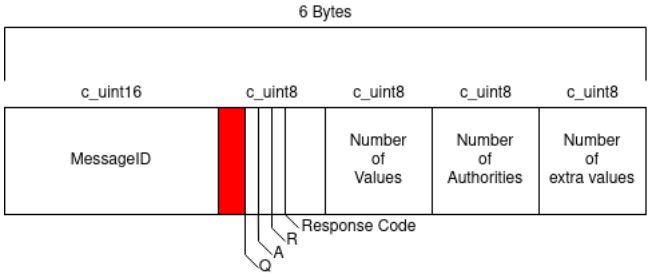

Figure 4.1: DNS header being used

We note that in this header we use the Python `ctypes` library for an ampler management and a bigger precision with the encoding and decoding of the header's information. This was greatly successful as there is one 16bit unit and four 8bit unit with even one of them being divided and not completely used, as indicated by the red region in the figure above.

4.2 Data Fields of the DNS

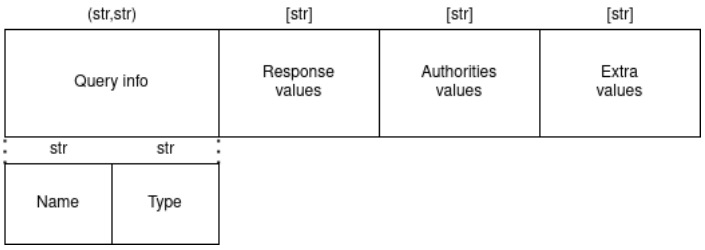

Figure 4.2: Data section of DNS PDU

4.3 Zone Transfer

As previously mentioned, the SS and SP communicate through TCP protocols. This happens in an exchange of packets. It begins in the SS as it send a query to verify its information validity, this is

done unlike the rest of this operation with the UDP protocol. The SP in response sends its data base's serial number. If it differs from the one contained in the SS (meaning the information was modified), the server asks for the new and updated version of the SP's DB. After noticing that the SS has the authorities to have the information it sends the entries from that domain.

4.4 Development notes

Being more used to programming in C than in Python programming, a very straight forward way of implementing such a header in the C programming language would be with an `uint16_t` for the *message ID* and `uint8_t` for the subsequent fields.

As is known, the real implementation of a DNS system would probably be in another language, like C or C++, but since we chose Python for this particular project, `ctypes` came as an interesting feature of the language that allowed us to have more manoeuvrability over efficient memory management.

5 Testing Ambient

For this first phase of the project we decided to build a simpler testing stage which we built on core (represented on the Figure 2.2). This version is built solely by one primary and one secondary server linked together and to the client by a switch.

6 Conclusions and future work

Our previous experience with Python At this first fase of reviewing the problem presented to us, we feel that the project is in the perfect direction, with a great planning for the future. We have a solid implementation of the servers. However, we recognize there is still a path to be built and there's some improvements to be made to minor issues. We are happy to present this project at this early stage and we hope that it continues to grow as we intent.

Bibliography

[1] P. Mockapetris, *"RFC 1035: Domain names - implementation and specification", Document search and retrieval page, 1987*,
https://www.rfc-editor.org/rfc/rfc1035
Last accessed 23 Nov 2022.

[2] P. Mockapetris, *"RFC 1034 - domain names - concepts and facilities", Document search and retrieval page, 1987*,
https://datatracker.ietf.org/doc/html/rfc1034
Last accessed 23 Nov 2022.

[3] M. Rey *"RFC 793: Transmission control protocol", Document search and retrieval page, 1981*,
https://www.rfc-editor.org/rfc/rfc793
Last accessed 23 Nov 2022.

[4] J. Postel *"RFC 768: User Datagram Protocol", Document search and retrieval page, 1980*,
https://www.rfc-editor.org/rfc/rfc768
Last accessed 23 Nov 2022.