uc3m | Universidad Carlos III de Madrid

University Degree in Telematics Engineering
Academic Year 2017-2018

Bachelor Thesis

"A Conversational Bot Expert in TCP/IP"

Luis Félix González Blázquez

Tutor

Ignacio Soto Campos

Madrid, 2018

ABSTRACT

When studying a telecommunication degree, it can be sometimes hard to remember all concepts or memorizing in detail how certain protocols work. To answer this problem, this project aimed to study how to create a bot in order to answer simple questions regarding the TCP/IP protocols.

First of all, it was necessary to analyse general information about conversational bots and programming tools in order to choose how to make the best implementation possible. Afterwards, we proposed different design alternatives that had to be done in order to develop the bot. These alternatives included the creation of a new algorithm to analyse text from users and obtain the main concepts for creating answers to questions. Finally, we divided TeCePe's implementation in programming modules that perform each of its functionalities separately to make easier its analysis and addition to the general code.

Users' results suggest that bots like TeCePe could provide some benefits to students while studying a subject. They usually prefer realistic human interactions and want more additional features besides bot's main functionality in order to be encouraged to use conversational bots, which are not very popular in the education field at this moment.

The main results of this project are generally favourable, as the bot developed fulfilled most requirements using all algorithms proposed. TeCePe is fast when searching, and can correctly detect users' intention in order to output the best possible answer.

Key words: Internet Bots, Natural language processing, Telegram, Python.

RESUMEN

Al estudiar un grado en ingeniería de telecomunicaciones, puede ocurrir que sea difícil recordar todos los conceptos dados en clase o memorizar cómo funcionan algunos protocolos. Para resolver este problema, en este proyecto se ha estudiado como crear un bot para resolver preguntas sencillas relacionadas con los protocolos TCP/IP.

En primer lugar fue necesario un análisis sobre los bots conversacionales y herramientas de programación para poder realizar la mejor implementación posible. A continuación se propusieron diferentes alternativas de diseño que deberían realizarse para desarrollar el bot. Estas alternativas incluyen crear un nuevo algoritmo para analizar textos de los usuarios y obtener los principales conceptos e ideas para crear las respuestas del bot. Por último, dividimos la implementación de TeCePe en diferentes módulos de programación, realizando cada una de las funciones de TeCePe por separado para hacer la programación más sencilla y facilitar su integración con el código principal.

Los resultados con usuarios sugieren que bots como TeCePe podrían otorgar algunos beneficios a los estudiantes que estén estudiando una asignatura concreta. Normalmente prefieren interacciones realistas (similares a las humanas) y quieren funcionalidades extra para que estén motivados a utilizar bots conversacionales, que no son muy populares en el campo educativo por el momento.

Los principales resultados del proyecto son generalmente favorables, puesto que el bot desarrollado cumple la mayoría de los requisitos utilizando todos los algoritmos propuestos anteriormente. TeCePe es rápido en sus búsquedas y puede detectar las intenciones de los usuarios para dar la mejor respuesta posible en cada caso.

SPECIAL THANKS

I would like to thank everyone that has been with me during the duration of the project, giving me support and cheering me to pursue my dreams.

Thank you Günter for supporting me all this years. Without you and your constant smile, this project would have been much more difficult.

Thank you Ignacio, my bachelor thesis tutor, for all you guidance and patience. This project would not be half of what it is now without your instructions.

Thank you Aveiro for opening my eyes to the world around me. I am who I want to be because you taught me so many valuable lessons I will carry with me for the rest of my life.

Thank you Jue, my dearest love, for all the time we have been together. Distance or time does not matter when I know you support me and make me feel each day like I am by your side. Soon the wait will be over, and I cannot wait for that day.

Finally, I would like to thank my parents for everything I have got in my life. Without your unconditional support, I would not have gotten this far. Thank you so much.

INDEX

IMAGE INDEX

TABLE INDEX

1. INTRODUCTION

1.1. Project's motivation

In modern society, technology has evolved up to a point where it has taken a major role in our daily life. We use it for doing almost everything to fulfil all needs we might have at every moment: ordering products online through Amazon, playing our favourite videogames, watching series online… Moreover, smartphones have gone one step beyond by revolutionizing human relationships in various unsuspected, yet interesting, ways.

Precisely, one of computer science main developments has been the creation of artificial intelligences capable of imitating human behaviours with the maximum possible fidelity. Many companies are trying to incorporate these technologies to increase productivity or add extra functionalities to their products. One of those technologies is the so-called Internet bots, which are computer programs that relieve the amount of work that has to be done when performing tedious tasks. There are many examples, but one of the most well-known ones is SIRI, the iPhone assistant. This bot has the capacity to hear and interpret simple commands from users to perform different actions on the phone (making phone calls, send text messages). Furthermore, it can also make small conversations with users. This last functionality is one of the most interesting ones it has, as imitating human speech is a very interesting concept.

Although these technologies have started to grow in popularity in the last couple of years, it seems that they are underdeveloped in certain areas if we imagine the limitless potential they have. One of those fields, where applying these technologies could be very beneficial is education. Particularly, they could be used to provide various powerful tools that would help students to further develop their knowledge and get more invested inside their degrees.

Hence, the idea of this project was conceived. TeCePe is a bot that has the aim of help students in telecommunication related degrees from all over the world to study the TCP/IP protocols. This bot provides a friendly chat-based interface to reply questions about TCP/IP protocols, it is fast and efficient, and all its information must come from a trusted source (Kurose & Ross, 2012) in order to reach as many people as possible.

1.2. Objectives

The main objective of this project is designing and developing a conversational bot to support users interested in learning about the TCP/IP protocols. To be more precise, the functionality of the bot is the following:

- The bot must provide definitions of basic concepts about the TCP/IP protocol.
- The bot must be able to compare two concepts of the TCP/IP protocol, as long as they are compatible.
- The bot must be able to interpret the user's intention to provide the best answer possible.
- The interface between the user and the bot must be clear and intuitive.
- The data to provide the answers must be stored in the most flexible way possible in order to simplify updates and modifications inside the data.
- The bot must interact as much as possible with users to simulate a human conversation and encourage the interaction with the bot.
- The bot must be able to deal with user's spelling mistakes.

Moreover, due to the nature of this project, we also have certain sub-objectives regarding project's development that must be fulfilled to. These ones are:

- Proposing some algorithms and tools to create TeCePe's functionalities.
- Investigate alternatives for implementing the bot.
- Make verification tests with users.

1.3. Document structure

This document is divided in the following chapters, including three annexes:

- Introduction: This chapter describes the main objectives of the bot and provides document's structure
- State of the art: This chapter describes the main characteristics of Internet bots and chatbots, including its classification, main technologies used and description of some examples. Furthermore, some general programming tools commonly used in bots are presented as well.
- Functional analysis of TeCePe: This chapter presents a general idea of the basic bot's functioning as well as detailing all functionalities required and used algorithms.
- Implementation of the solution: This chapter presents the technical solution used to implement TeCePe, as well as describing in detail all libraries used during its development.
- Verification tests: This chapter exposes the results obtained throughout TeCePe's testing phases, both local and user.
- Project's planning: This chapter shows the general plan followed during TeCePe's development, including a table of activities and its corresponding

Gantt diagram, as well as economical details of the project. The economic environment is detailed inside this chapter

- Conclusions: This chapter summarizes the main conclusions of this project and proposes some improvements to be made in the future.
- Appendix A: This chapter includes the template of user's test phase used to carry out all tests with users.
- Appendix B: This chapter includes all current TeCePe's database content.
- Appendix C: This chapter describes how TeCePe must be deployed in the external server where it is hosted.
- Appendix D: This chapter describes the complete legal framework of TeCePe.

2. STATE OF THE ART

2.1. What are Internet bots?

A bot (or Internet bot), acronym for robot, is a piece of software that performs repetitive tasks automatically through the use of code, usually performing actions less efficiently done by a human (Symantec, n.d.). These tasks may vary depending on the type of bot used: tweeting at a certain time of the day, buying automatically a product once it is available on a website…

Internet bots are a growing trend that is quickly spreading every day. According to IMPERVA INCAPSULA's annual report, (Zeifman, 2017) 51.8% of total website visits were done by bots. Therefore, the number of bot types that exists throughout internet is almost unlimited.

There is not a strict "category" definition that suits all possible bots, so criteria may vary depending on the point of view of each study and organization. For example, Imperva proposed dividing them into professional and user bots, depending on who are the developers. Among the first ones, they can also be divided in beneficial and malicious bots, categorizing them depending on which task they perform. As seen on *Fig 3.1*, most beneficial Internet bots are usually feed fetcher bots (12.2%) and search engine bots (6.6%), while most malicious bots are impersonators (24.3%) and hacker tools (2.6%) (Zeifman, 2017).

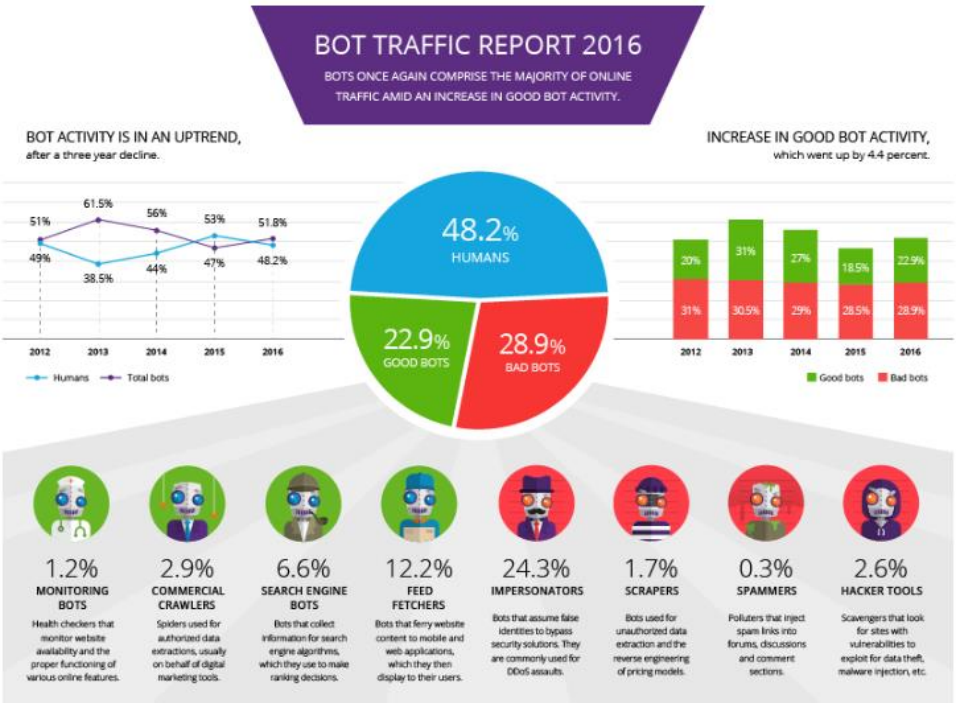

Fig. 2.1 2016 Imperva Anual Report (Zeifman, 2017)

Nevertheless, there are many other ways to classify bots. One of the most relevant, and formal, is classifying bots according to their implementation

Bots are created based on different purposes that developers have in mind: requirements and technologies used in the creative process will not be the same if the approach is different. For example, designing a conversational bot that will learn and refine its response algorithm based on user's input will not be the same as if the bot would just randomly choose answers without taking into account previous inputs: scopes are completely different, so development will be different on each case.

Following that criteria, bots tend to be classified in three main categories:

1. Stateless Bots: Do not keep track of user's information, so they will not vary its functioning regardless of previously received information. Although this may seem like a disadvantage at first glance, this property should not necessarily be regarded as such: the simpler the bot is, the easier is to program it and less resources and/or time will be required. Therefore, it is not surprising that most "user developed" bots fall into this category for its simplicity (Giridhar, 2017).
2. Stateful Bots: Stateful bots are capable of remembering information of all previous user iterations. Therefore, this type of bot is able to provide much more complex functionalities than stateless bots. For example, a conversational bot can ask influence users to change the topic of a conversation if the bot detects that it is redundant, or provide recommendations based on user's preferences. Nevertheless, these bots start to get much more complex to be developed, as its sophistication is greater than memoryless bots. (Giridhar, 2017)
3. Smart Bots: Also labelled as "learning bots", these do not just remember all information introduced before, but also use algorithms to predict user's future inputs and react accordingly. Some mechanisms used in this bots are machine learning, understanding of language semantics, prediction algorithms… They usually incorporate novel technologies and are capable of tasks too complex to be done without automation. Due to its great complexity and necessity to be updated constantly with the most modern technologies of the market, its development is extremely difficult. However, their possibilities are huge for creating new and innovative applications. (Giridhar, 2017)

It is worth mentioning that inside these categories exist also sub-classes that fall in between two implementations. For example, another subcategory could be "semi-stateful" bots, which have some limited memory capacity and, therefore, it is neither stateless (because it has memory), nor stateful (because it cannot remember all previous inputs) (Bunardzic, 2017).

2.2. Chatbots

A type of bot is the so-called “Conversational bot”, which is able to simulate a conversation with another human being, when in reality a bot. A bot that is able to chat as a human can have some advantages for users in many ways, like in the field of entertainment. Some of the most well-known applications include Cleverbot and Simsimi: very popular bots that try to perform a full length conversation for recreational purposes. They store all input answers and try to react to newer ones using all previous inputs from other users in past conversations (Wolchover, 2011).

Fig. 2.2 Conversation with Cleverbot (Carpenter, n.d.)

Regardless of how good their algorithm implementation is or how accurate the conversation is, Cleverbot is still very popular between bot Internet users, reaching 1.95 million views per month (SimilarWeb LTD, 2018), and developers for its ever-expanding API.

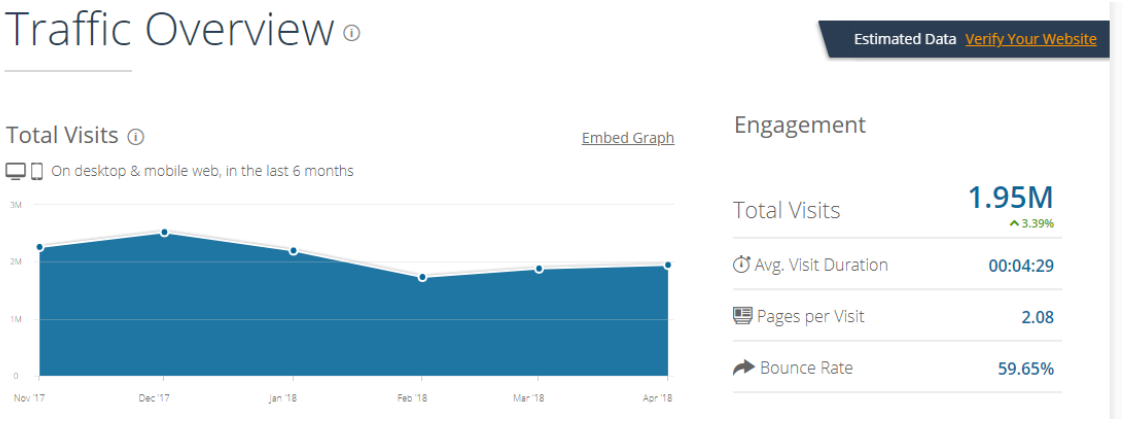

Fig. 2.3 Cleverbot web statistics (SimilarWeb, 2017)

Furthermore, Cleverbot is just a single example of the amount of non-professional bots around the internet. The number of detected bots accounts across Twitter has been reported to be approximately between a 9% to a 15% of its total user base; perhaps even higher due on how increasing bot's complexity is becoming, where sometimes is challenging to distinguish a bot account from a real user. (Onur, Ferrara, Davis, Menczer, & Flammini, 2017)

Moreover, companies have just started to realize the potential of conversational bots to provide clients with new ways to perform actions that would take more time through the use of traditional human interactions. Some advantages of bots for companies have could be, but are not limited to (Maruti Techlabs, 2017):

- Permanent availability
- Easier customer handling
- Personnel's cost reduction
- Personalized assistance

Depending on the type of conversation that a bot may have, conversational bots can be classified in these categories:

1. Spoken Dialogue Systems (SDS): Originally based on a model where users could communicate with automated systems via text, this type of conversational bot was created to provide services by using linguistic tools. However, it was continuously used in laboratories and research facilities to test how to develop interactions using more complex algorithms and discovering new technologies to improve them. They have continuously been used for research purposes, and the most advanced ones can be found in important universities and laboratories form all around the world (McTear, Callejas, & Griol, 2016).

2. Voice User Interface (VUI): This type of conversational bot was developed when people started to realise that voice could be used in order to manage simple tasks such as redirections to other services for increasing productivity. This type is widely used nowadays on companies to help redirecting different kind of calls to the correct departments by using voice commands (McTear, Callejas, & Griol, 2016).
3. Embodied Conversational Agents (ECA): This type of conversational bot tries to reach a full human (and conversational) experience by using different mechanisms involved in human talking: verbal communication as well as facial (and sometimes body) language. They are usually presented with a 3-D model that represents a character or a human. They are used to provide entertainment and add some social component to certain products/services. Furthermore, they have grown in popularity lately in the education field (McTear, Callejas, & Griol, 2016).
4. Pure conversational Chatbots: Possibly the most common of all (or most popular at the very least), this type of conversational bot tries to emulate a complete conversation to trick users into thinking that the bot is actually another human. The difference with the previous ones is that this bot is not meant to provide any service apart from the mere mimicking of human behaviour. Therefore, its internal mechanisms end up being less complex compared to previous ones. (McTear, Callejas, & Griol, 2016)

Chatbots are a growing trend in society for every type of user, providing a great amount of advantages to potential users, making them very attractive to be developed at this moment of time.

2.3. Educational bots

One field where conversational bots have started to grow in popularity is the field of education: these bots can potentially provide useful tools for teachers and students in many different ways. Some examples may include the possibility of giving feedback for a subject, classes, laboratories…, creating online tests for helping/testing student's knowledge, and providing different mechanisms to help reinforcing more traditional studying methods.

Obtaining data about educational-centred bots becomes a more difficult task than in previous cases though, as these are usually applications developed by individuals, or belong to private education networks where tracking statistics is far more difficult.

Designing an educational-centred bot comes with some additional challenges that have to be taken into account:

1. Motivation: It is important to have in mind that bots must be an improvement for both teachers and students, not adding more difficulties to the learning process.
2. Difficulty: Programming a bot might require some researching and developing some skills that may not be possessed for some individuals and, in consequence, would provide an extra layer of difficulty that might affect teacher's performance, or even dissuade of developing the bot.
3. Time: Developing a bot requires some time to design, program and maintain the bot.

Educational chatbots can be a powerful tool for everyone inside the education system. However, before creating a bot of these characteristics, some considerations have to be taken into account.

2.4. Artificial Intelligence

One of the most developed technologies in recent years has been artificial intelligence, which is an area of computer sciences that emphasizes creation of intelligent machines that work, and react, like humans (Technopedia Inc., n.d.). This term in reality is vague, as AI covers many topics and sectors, so it cannot be summed up in a single sentence. However, it is possible to classify AI depending on the quality of the algorithms and complexity of the tasks that any machine or program performs. "Strong" AI would be the one that has a similar, or superior, intelligence to humans; capable of performing actions that would rather be too complex for machines, involving concepts like self-awareness or consciousness. "Weak" AI would be those that only serve specific tasks, not having enough capacity for doing anything else than what the bot was programmed for. Chatbots would fall inside this category, as they are not designed to do anything else but simulating conversations with users (McNeal & NewYear, 2013).

It is important to remark that AI is not always related to machines' learning process. In other words, even though a machine or program might not learn from user inputs, if its technology tries to emulate human-like behaviour, or makes specific tasks that resemble human actions, then it can be considered artificial intelligence.

In the case of chatbots, one of the most used tools to produce this AI behaviour is Natural Language Processing.

2.5. Natural Language Processing

Natural Language Processing, or NPL, can be defined as how computers understand and manipulate natural language data to perform certain actions (Chowdhury, 2018). In theory, the main goal of NLP is having a full Natural Language Understanding (NLU), which should suffice to achieve the following objectives in a perfect system:

1. Paraphrase an input text
2. Translate text into another language
3. Answer questions about text's content
4. Draw inferences from text

While a lot of progress has been made with objectives one to three, the last one is still one of the main challenges of these technologies, as systems still find hard to draw inferences from text. (Liddy, 2001)

Nevertheless, the objectives to pursue vary depending on each application. In chatbots, the three first objectives can be considered as ideal, as they provide enough means to extract information from sentences, translate text and formulate a valid response.

2.6. Examples of chatbots

One of the most famous chatbots is the ELIZA project. Created in 1966, this chatbot was one of the first projects of its kind, although it spawned as a parody of the responses of a psychotherapist in an initial psychiatric interview. ELIZA basically detected common words in human speech and put them inside previously created templates. This technology was praised for its believability (at the time), even to the point of starting a new debate about Artificial Intelligence limits (Deryugina, 2010).

Almost in parallel, PARRY was developed in 1972 by the psychiatrist Kenneth Colby to emulate a paranoid schizophrenic patient. Instead of using key words, PARRY "implements a crude line of a conversation" (Deryugina, 2010). Both technologies were at the peak of chatbot development and even their dialogue through ARPANET is considered one of the most important technological goals of the decade (IETF, 1973) .

All subsequent bots have been usually developed based on algorithms settled by those two conversational bots. The main "modern" conversational bots are Albert One and ALICE, which both won Loebner Prize's at least once. This prize is one of the most AI prestigious awards any bot can obtain, as it is the oldest Turing Test contest in the world (AISB, 2018).

Albert One is a conversational bot created by Robby Garner based on the ELIZA project. During the first phases of development, a huge compilation of things that people would say to a chatbot was created and stored inside a database. These responses

were the basis of all answers that the bot would output to users. This data, combined with a series of hierarchical subsystems including a portion of ELIZA's program, was enough to create a bot capable of winning the contest in 1998 and 1999.

ALICE (Artificial Linguistic Internet Computer Entity) is one of the most famous conversational bots in history. Inspired by ELIZA, ALICE processes inputs similarly, but adding a framework that can be implemented and modified to model human dialogue in limited domains. It was implemented using artificial intelligence markup language (AIML), a language specifically created for ALICE by Dr. Richard S. Wallace, the same creator of the bot. (Holtgraves & Han, 2007)

Regarding educational bots, one of the most well-known in recent history has been Jill Watson. Developed in the Georgia Technological University, this bot simulated to be one of university's teachers for approximately six months, without anyone noticing that she was in reality a bot. This bot showcased how great conversational bots' potential could be inside the education field (Michael, 2016).

2.7. Bot development

Developers have been getting new more advanced software technologies and tools for developing bots at an exponential rate, and their accessibility has never been easier. There are many different ways to develop a bot, and it has become easier for users with a relatively low knowledge in programming and/or computer fields.

APIs and frameworks

There are many platforms and applications that support bot development. Most of these platforms have different libraries and APIs that ease programing for developers. However, what is exactly an API and how does it help creators? An API (acronym of Application Programming Interfaces) is a set of functions or standards that make easier the communication between software elements of the system, mainly between the subroutines of the operative system and the program itself. (MuleSoft Inc., 2015)

However, in most cases, APIs are not the only programmers' facility source. Some applications usually provide a framework to work with, which essentially is written code by users or organizations to provide some functionality to make tasks simpler within the language/application that would otherwise be tedious to make for the programmer. It is also worth mentioning that, due to the increasing popularity of bot development, these frameworks are usually created by different companies in order to let common users build their own personal bots using their applications. Not all frameworks are the same, as some of them provide functionalities that others do not, so they must be taken into account when deciding which one should be used.

Different frameworks require different programming languages with their own set of rules and specifications. This last point is far from trivial, as choosing the most suitable language for bot development can affect heavily the quality of a bot. For example, one of the most popular frameworks for web developing in CSS is twitter's bootstrap, while others use Zurb's Foundation. Nevertheless, a lot of modern frameworks rely on one programming language: PYTHON.

PYTHON is an Object-Oriented programming language fairly similar to Java and C++ (Python Software Foundation, 2001). However, PYTHON has some powerful advantages over those two that makes it a very powerful language for coding, especially for users that start coding for the first time. Firstly, PYTHON is an easy-to-pick language because a lot of "traditional" elements in programming languages have been reworked in favour of a simpler structure that makes understanding code easier. Secondly, Python relies heavily on external libraries, which add lots of functionalities that would be tedious and/or complex if had to be written manually. Some examples of these libraries could be the NUMPHY library, which adds to PYTHON some MATLAB functionalities, turning the language in a very useful tool for mathematical analysis or simple networks simulation; or the TWEETPY library, which helps users to have a much simpler bot development in Twitter by simplifying tasks like connection to feed, authentication functions, etc…

Many more tools users might want to implement vary depending on bot's complexity. Some considerations have to be taken into account: will the bot store data? If so, where will the bot store it? What formats will be used to do so? Does the bot require reading some external data? Solutions will vary depending on the approach that developers decide to take: in some cases using external structures outside the application might be useful (JSON) and other times using databases might be a better way to read/store data. Furthermore, these considerations only take into account data manipulation; bots evolve at all times, increasing its complexity and functionalities that they can provide to final users. Fortunately, due to the recent improvement of technologies that allows communication between users and devices, the possibilities grow each day with the creation of new tools such as LUIS (Language Understanding Intelligent Service), which allows applications to understand user's particular speech based on their own words (Microsoft, n.d.), or Speech APIS that give bots some voice to communicate with users.

3. FUNCTIONAL ANALYSIS OF TECEPE

3.1. Introduction

In this chapter we are going to describe how TeCePe's general structure is, and describe in detail all strategies taken for every aspect of TeCePe's internal functioning.

3.2. Objective

Studying certain engineering subjects or/and concepts can sometimes be really challenging for most students. When they are given for the first time, students might struggle remembering exactly what the main ideas of the subject are. In telecommunication engineering, specifically in the TCP/IP protocols, this problem can be even more challenging, as information found throughout Internet is not always a reliable source of information due to telecommunication's great complexity. It is true that students usually have references of books where they can get all needed support. However, these books also pose some challenges for students, as they depend on its availability when no electronic format is supported, or can be written in not user-friendly formats that are not easy to read. This might be frustrating for some students when the problem that needs to be addressed is very simple and would only require a small search (which could be more time consuming if it needs to be localized inside a book).

To solve this problem, a portable encyclopaedia could be developed, which would allow users to solve any doubt by searching quickly any concept from a given subject at any moment. Moreover, its interface could have an attractive style that would encourage users to interact with it. That is what we have developed with TeCePe.

TeCePe is a conversational bot that allows users to ask questions about the TCP and IP protocols. The answers are based on the book "Computer Networking: a top-down approach" (Kurose & Ross, 2012). This conversational bot has to be quick in providing coherent and concise answers while presenting information in the most attractive way possible through a simple, yet attractive, interface. The targeted audience of this project are telecommunication engineering students and people interested in the TCP/IP protocols.

3.3. High-level description

TeCePe is a conversational bot. Therefore, its functioning is fairly similar to a standard chatbot, with the exception that answers make particular replies in a concrete domain rather than mimicking a full-length conversation.

Users access the application through a mobile phone or a PC. On its interface, they should only see previous answers (if they have any) and an input text-box, similarly to other conversational applications like Whatsapp or Telegram, to give the feeling of talking with another user. An example of this interface could be the one shown in *Fig 3.1*:

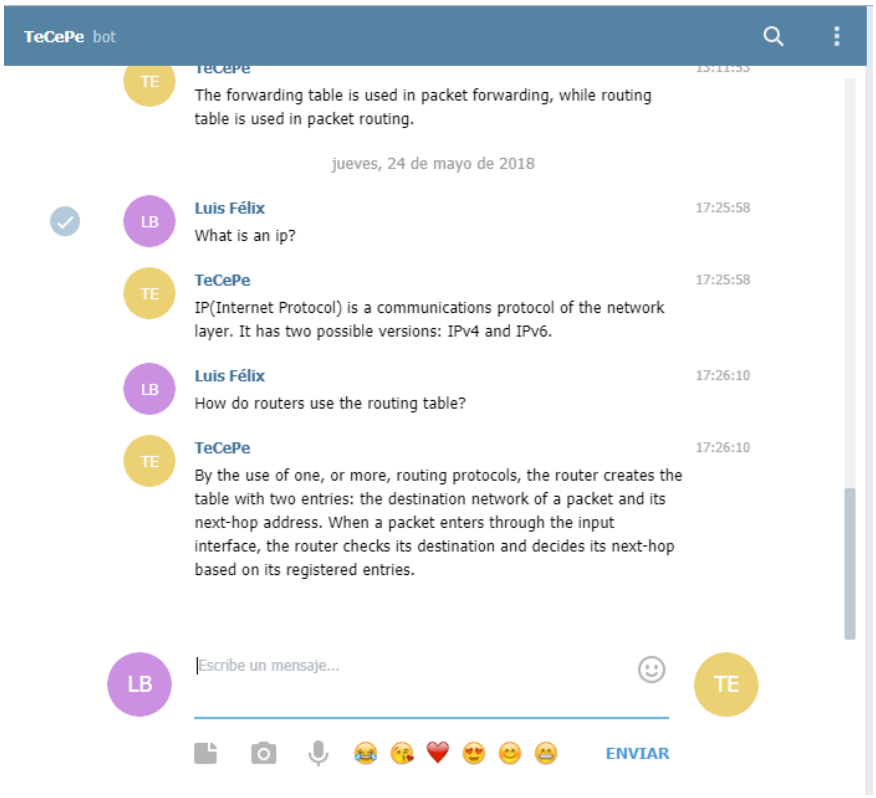

Fig. 3.1 Telegram Interface

In *Fig. 3.2* we can see all interactions between bot's components with their correspondent steps performed every time a search is performed:

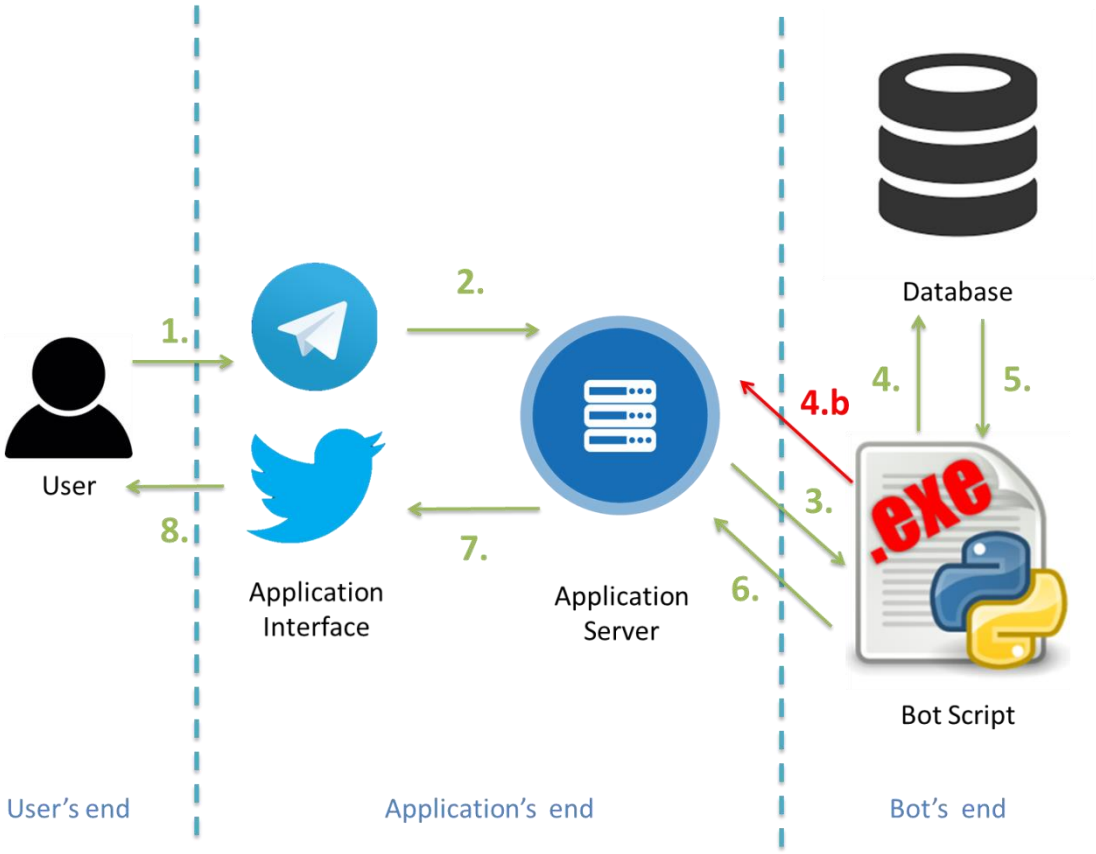

Fig. 3.2 How TeCePe creates an answer

1. The user introduces an input text on the application's text-box and presses the send button when done.
2. Data is transmitted from the application interface to the front-end application's server.
3. The server sends to TeCePe the information to perform the front-end to back-end communication.
4. TeCePe retrieves this text and extracts the relevant information from the input sentence that will be used to create an answer. Afterwards, it will contact a database to search the requested answer.

b. If no information is found inside the sentence, an error message will be created and outputted to the user. Therefore, skip to step 7.

5. Database replies with the requested answer.
6. TeCePe adds any necessary details to the answer and sends it back to application's server.
7. The application's server outputs this message to the interface.
8. The final answer is shown as a text message to the user.

This is simply a high-level model that showcases the basic TeCePe's functionality. In order to create this application however, it is mandatory to use and/or develop a set of more concrete algorithms to provide our bot with the "intelligence" necessary to perform all required connections and analysis. These will be explained in later sections of this document.

3.4. Concept-analysis: priority keyword algorithm

Keyword searching approach

One of TeCePe's core functionalities is identifying and interpreting concepts from user's input.

The first design approach for this task could be using the keyword searching algorithm. It is the most natural method that comes to mind when looking at TeCePe's characteristics. As this bot is only interested in a concrete subject, detecting ideas is bot's first priority, regardless of sentence shape. This behaviour can easily be seen with this example:

1. *TeCePe, please tell me what IP is.*
2. *Hey, I would like to know if you can search what IP means, because I want to know.*

Both sentences ask exactly the same question to the bot, despite user's style of writing or the amount of words used; for TeCePe, it is only important to answer the definition of IP.

Users want what they asked for. Although it is true that providing a more elaborated and/or personal answer can improve bot's perception, if the key function, which is the search, is faulty, then the bot will not fulfil its purpose and could be considered a failure. Therefore, the first bot version of TeCePe initially implemented a simple "key-word" searching engine, which would compare its input with some fixed words and, if present, would output its definition. Otherwise, TeCePe would simply reply negatively, stating that she was not able to find the requested definition.

Hence it could be wrongly concluded that the algorithm was correct and the approach was right, when in reality, it has several limitations that must be addressed:

1. It can only detect the first keyword from the user's input, which will lead to a severe loss of information if the user requires more information.
2. It does not respect priorities between words.

This last problem is the main flaw of this solution, as the first one could be avoided to some extent. In common speech, when asking for definitions, usually the important part of sentence's global meaning is detected through the most concrete word. For example, if the required definition is "IP routing", then user's intention is obtaining a "routing" definition, asked using the word "routing". Furthermore, some ideas are implicit given the appropriate context and constraint (routing is indeed related to the IP protocol, so there is no need to specify the IP part). It is not always the case however, as sometimes preceding details are mandatory to have the complete request. For example, in the "routing table" case, the word "table" is not enough to have concept's full meaning. Nevertheless, it does not change the fact that still remains a case of priorities among words. With "key-word" searching though, it is impossible to provide this priority behaviour, thus another approach has to be explored.

This does not mean keyword searching is useless for the bot. Its simplicity makes it a perfect alternative for functions like word filtering. TeCePe does not want users to say bad words when searching for information. Therefore, it is not important where those words are located or how many are present: as long as they are present, TeCePe can refuse to answer the question if a simple algorithm run detected some bad words. It is also a good starting point to develop a more elaborated solution.

Priority key-word searching algorithm

Taking into account all requirements previously mentioned, we came up with an algorithm that allows using words with different priority levels after performing a search inside user's input sentence. It is a variant of the "key-word" searching algorithm: the priority "key-word" searching.

Naturally, the best structure to apply this type of algorithm must be a tree. That way, all nodes will have the required information while maintaining an order between components and separate in different priority levels. An example of this structure could be the one shown on *Fig 3.3*:

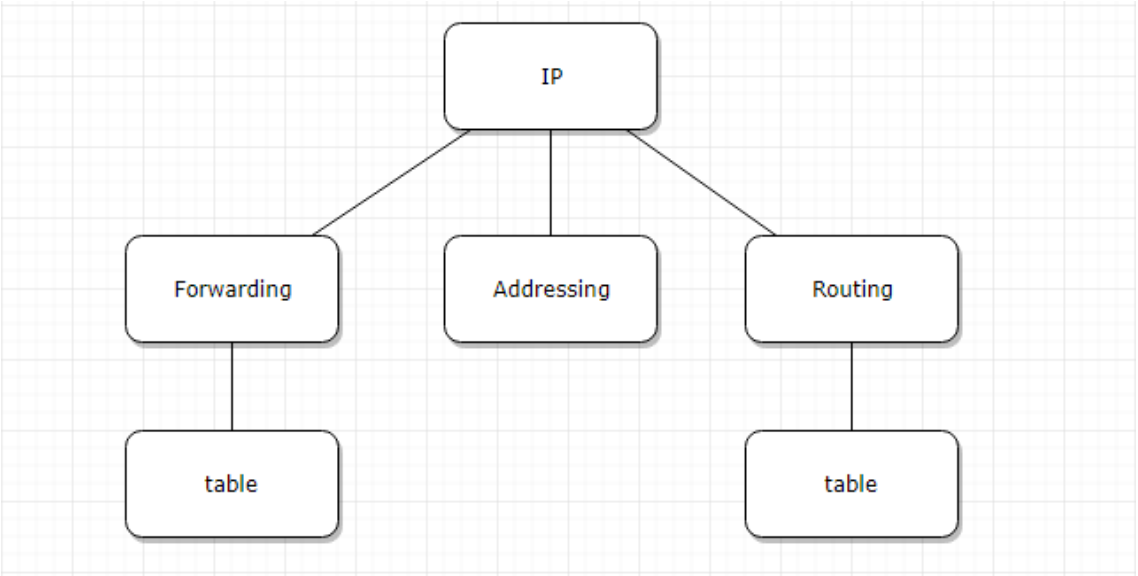

Fig. 3.3 Tree structure

In this tree we are able to see how all elements are arranged. All nodes contain the words users will use inside sentences, with the initial node being the subject's main covered topic. The rest of nodes are more concrete words related to each previous entry, creating priority levels among these words. The deeper each node is the higher priority it will have against the rest of words. This way we now have a format with all elements required, ordered by specification levels and stored inside a search tree.

However, before specifying how the algorithm runs, it is important to define the limitations that the bot has regarding its information handling plus some reasoning behind all of them. The main constrains TeCePe has are:

1. Only one concept can be asked each time, and comparisons between words can only be done in pairs. There are a couple of reasons behind this constrain, which are improving search efficiency (repeating the algorithm as less as possible) and that similar programs use this same approach to provide a more believable conversation between the user and bot.
2. Comparisons cannot be done between concepts of different levels: It is very infrequent that two elements without the same "level" of concretization can be compared (for example, comparing IP with routing table is impossible, as they have nothing in common). Although it can be argued that this is subjective between different users, the one that controls this validity is the data manager. Therefore, this decision is not part of the algorithm itself. Moreover, both the structure and algorithm must have tools to manipulate this constrain easily, so in case a modification is required, there should not be a problem.

In order to check better how the algorithm works, we can also use the previous tree to provide an example of its behaviour. Some possible inputs will be proposed and, afterwards, we will check how the algorithm runs with some visual help on that same tree:

1. *What is IP?*

In this case, the search would locate the word IP as its search input, and look inside the tree for that node. Once localized, the program will continue based on this decision.

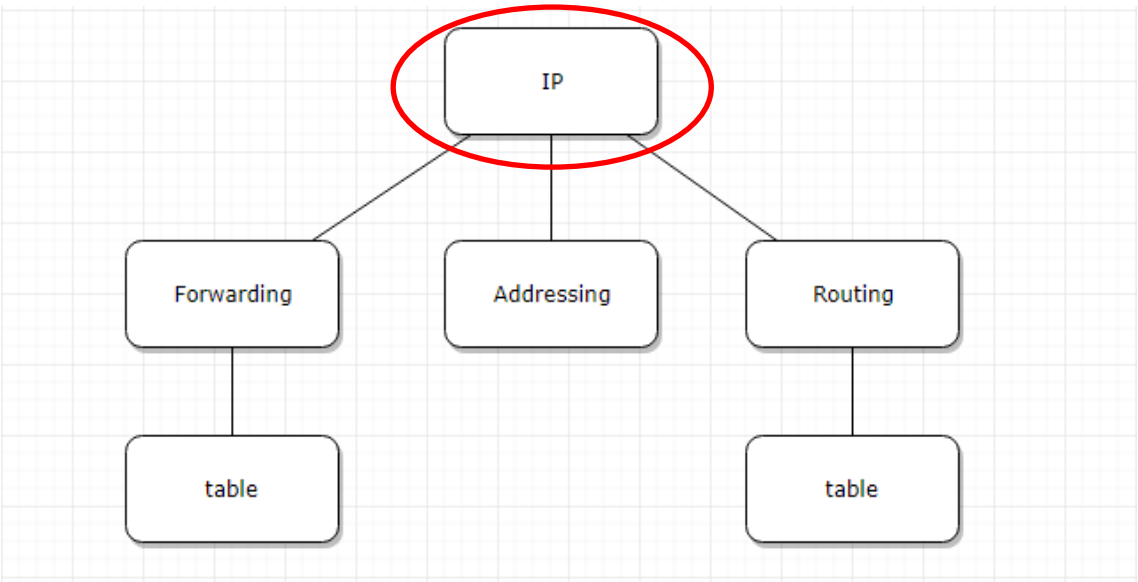

Fig. 3.4 IP retrieval

2. *What is IP routing?*

In this case, TeCePe would detect two elements to search inside the tree: IP and routing. Therefore, TeCePe would start localizing both nodes inside:

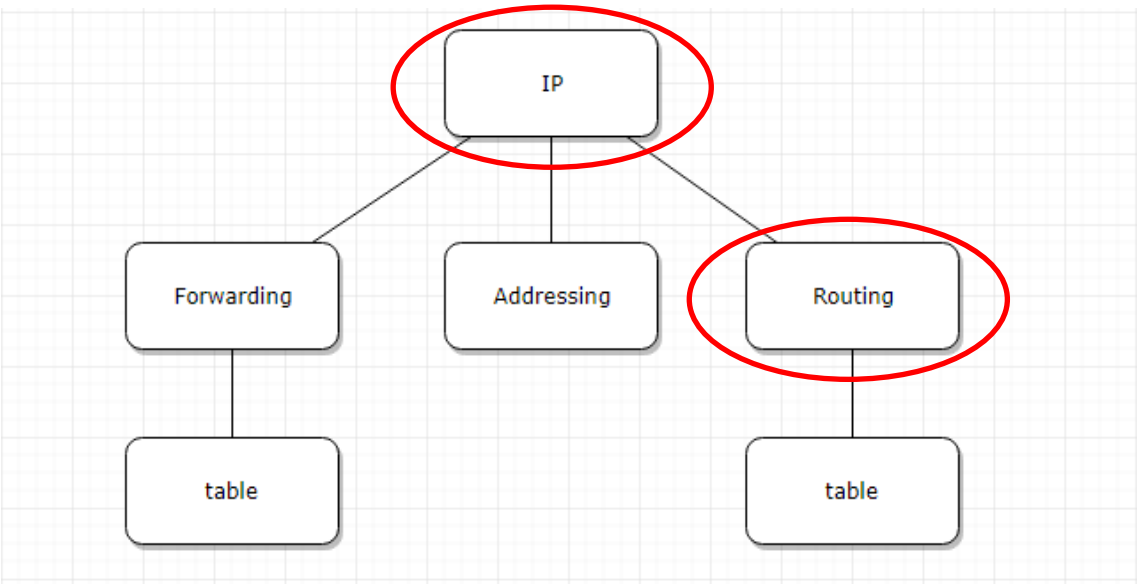

Fig. 3.5 IP and routing retrieval

The next step would be searching the deepness of both nodes and keeping the biggest one, which provides the most concrete definition inside the sentence:

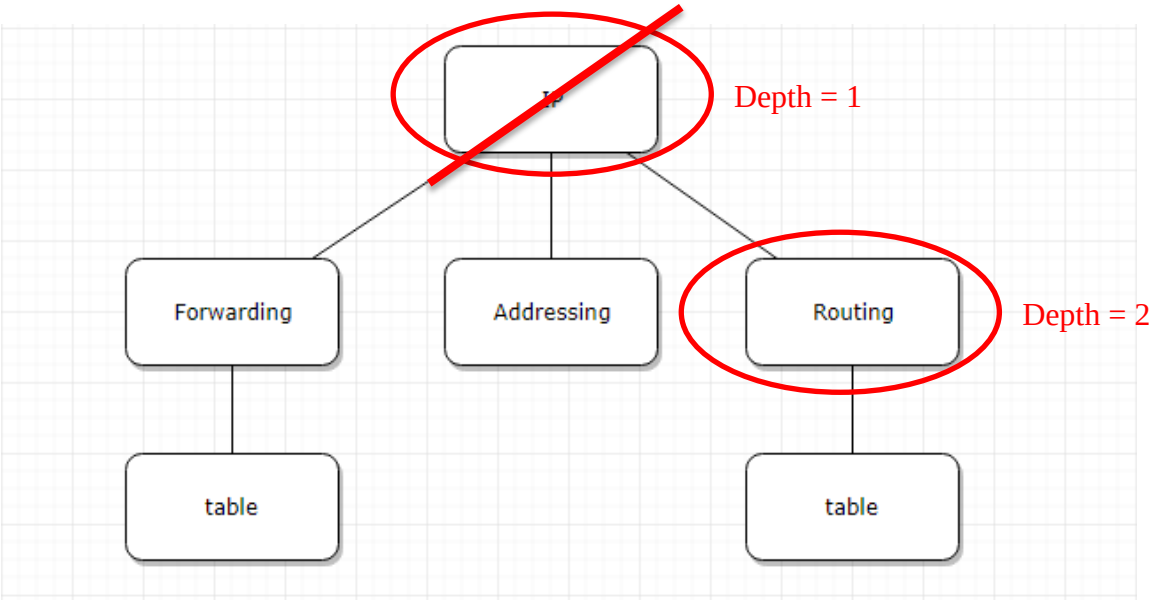

Fig. 3.6 Routing selection

Hence, the search's result would be the "routing" node.

Most tree nodes are self-defined: they do not require any previous unit to be selected, as context already references the previous node:

3. *What is routing?*

This similar sentence only has routing as its search unit, which would be a straightforward localization, as showcased in example 1, without the need of searching its depth.

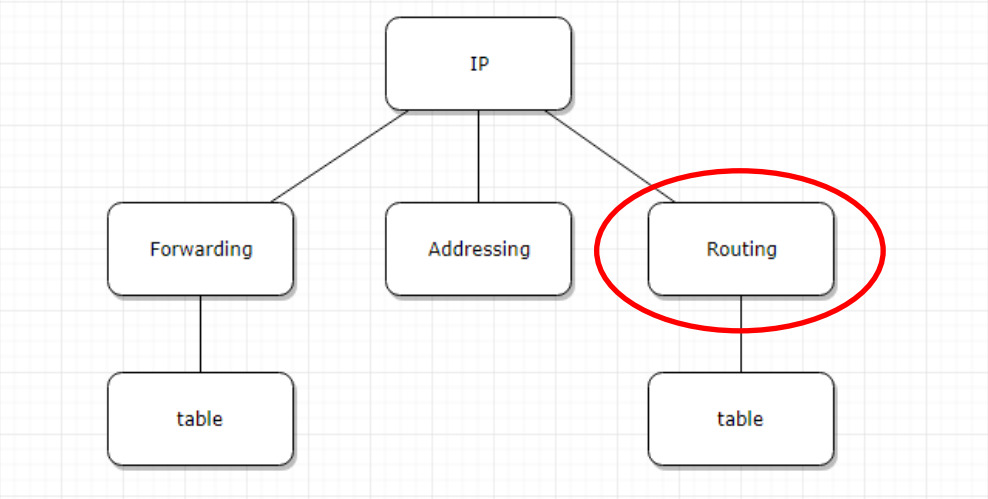

Fig. 3.7 Routing retrieval

The only nodes that do not satisfy this property are those that share the same name with others inside the tree:

4. *What is the table?*

In this case, the algorithm will locate two elements with the same depth. Usually this should be an error, as two concepts cannot share the same definition or output two different definitions:

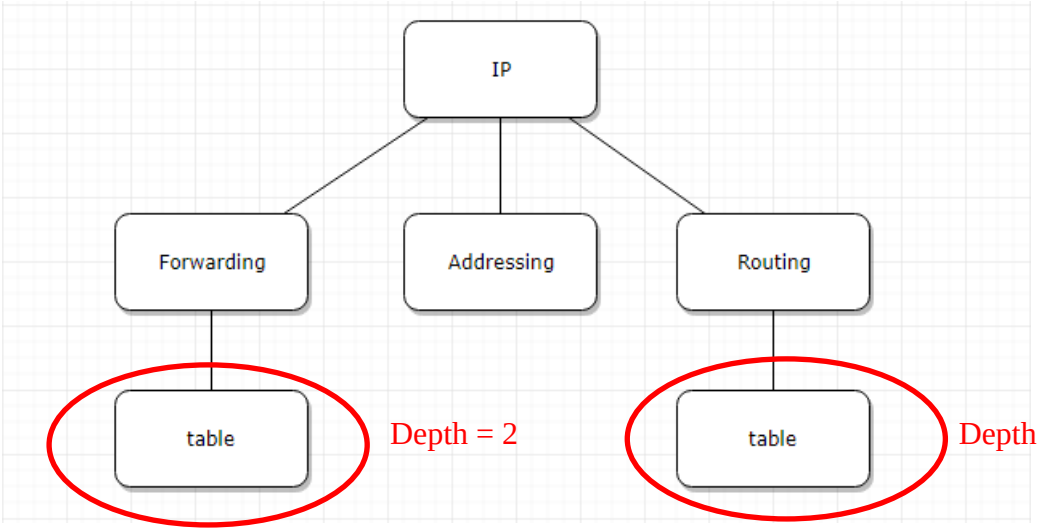

Fig. 3.8 Table retrieval tie

However, if we introduce a new concept like routing inside the previous sentence, then the algorithm is capable of being run again to check if a previous one is present and, by locating the word, selecting the correct node:

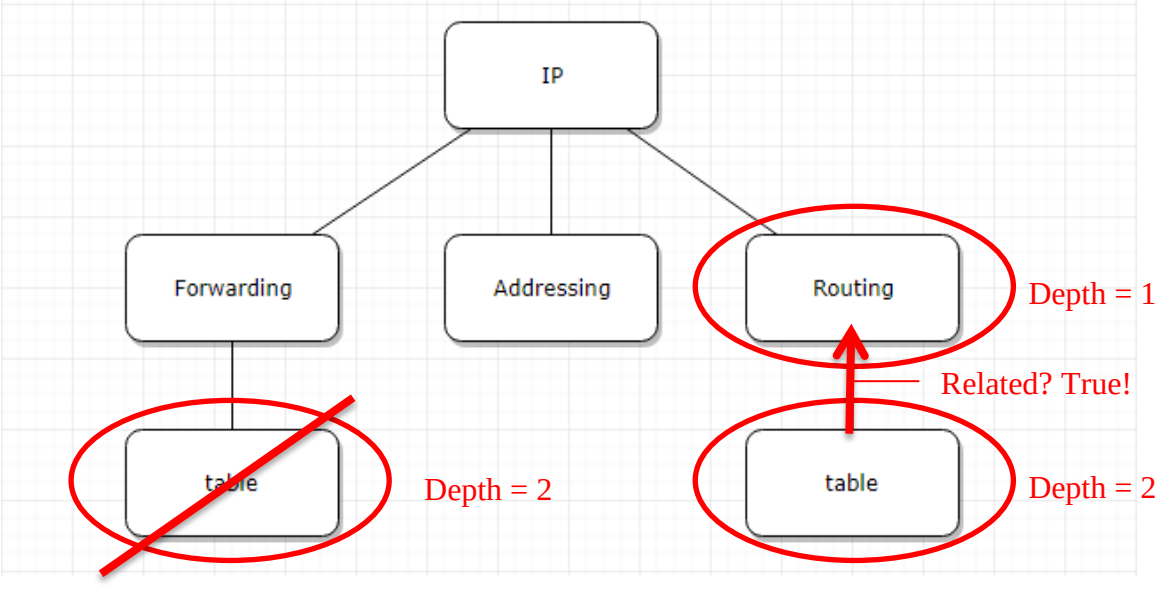

Fig. 3.9 Table selection

The result of applying the algorithm will be adding the previous node name to the original one.

Some words might also have a similar problem where ideas require additional words in order to make sense. For example, “distance vector” is composed of two words. Although there are no nodes per se, both words are required in order to form a complete definition. Otherwise, it can confuse users if only one word is enough to perform the

search, especially when the words can be used in other contexts. Hence, we used the same strategy as in the previous example to solve this problem:

5. *TeCePe, what is a routing protocol?*

In this new example, we will suppose routing requires the word “protocol” to be a valid search. The algorithm will find the routing node and, afterwards, perform the search to locate the “protocol” word inside the sentence, which is stored in the tree node:

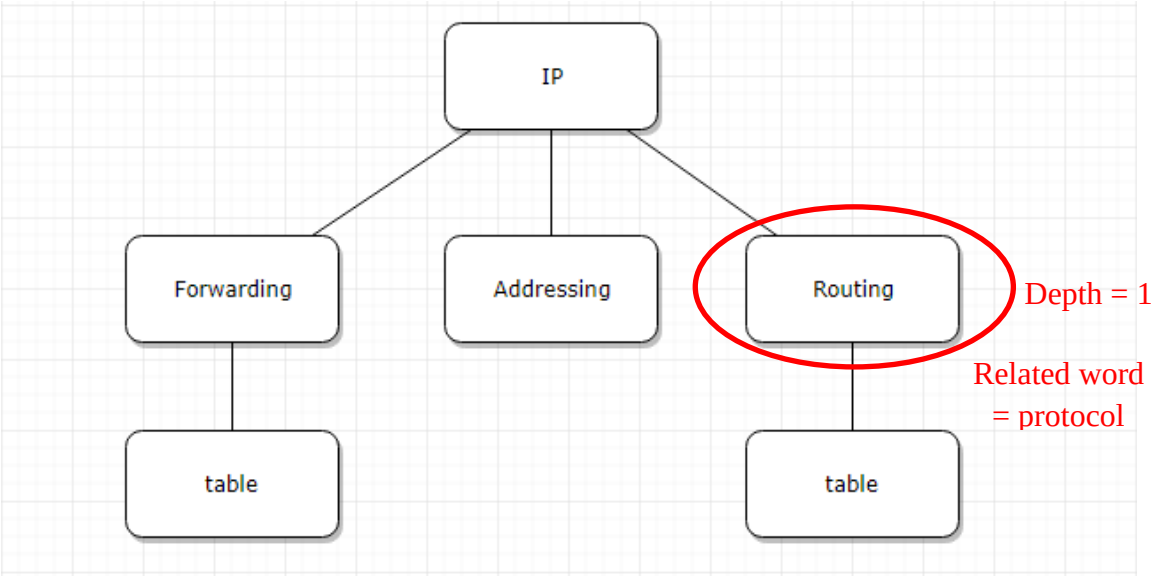

Fig. 3.10 Routing with protocol constrain

Thus, the algorithm final value would be the addition of these two words.

6. *TeCePe, what is Forwarding and routing?*

In this case, the algorithm would detect both nodes inside the tree and check if both have the same depth, so both units would be valid. However, two concepts cannot be outputted, so other criteria must be chosen to solve this conflict. We will see more of this behaviour in the following sub-chapter.

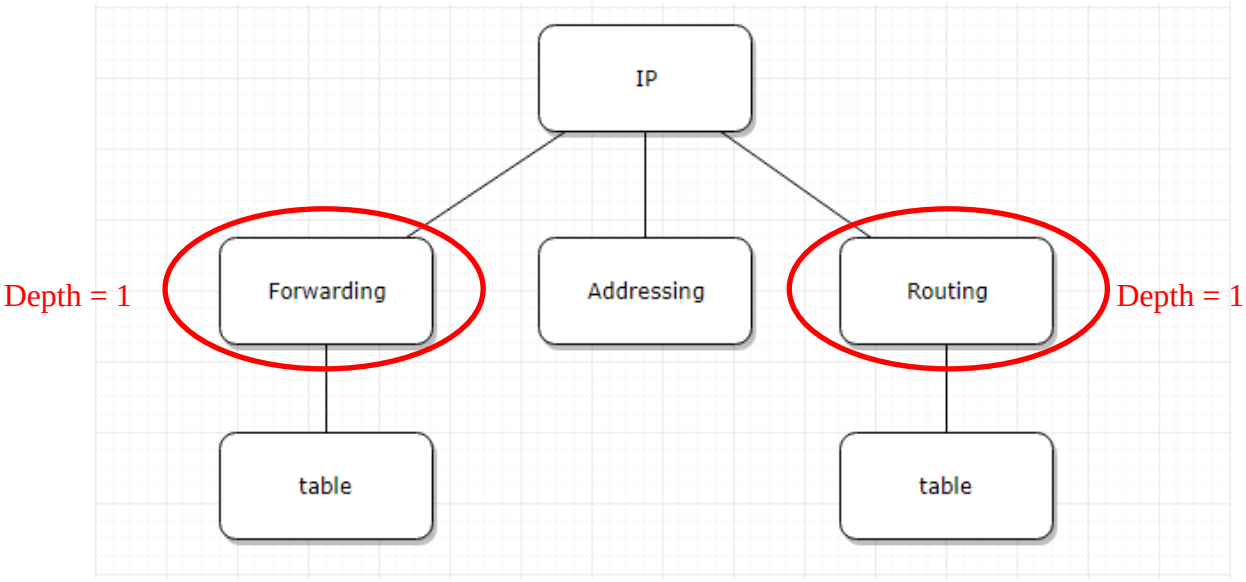

Fig. 3.11 Forwarding and routing tie

With this algorithm, we are capable of obtaining efficiently every concept required for searches inside the given tree. Therefore, this approach was the one used in TeCePe's development for its simplicity and efficiency.

Context matters

We know how to extract information related to every element that users want to ask about. However, this information is still very limited to provide a good quality response, as the actual "action" asked for is still not clear for TeCePe.

Usually, sentences are not only defined with just nouns or verbs. Most of the times, some adverb or other verbs are mandatory to get the full meaning of a sentence. For example, a sentence consisting only on "TeCePe, routing table" is very vague with its meaning, as it is impossible to clearly define the user's intention. Nevertheless, if the user asks: "*How* does the routing table work?", then it is possible to precise what the user requested. Moreover, by only extracting the elements "How" and "routing table", we can still have enough information to provide a coherent response.

Therefore, another search for sentence "context" is required. In this case, a simple keyword search is enough to obtain this element to create the full query that would be launched to the database.

But first, it is mandatory to define those required words. The most important units to be considered must be the adverbs: what, when, how and where. For comparisons, it will be necessary to select some verbs and nouns that express this intention, like compare, comparison, difference… The rest of words will be decided by the data manager, as they are dependent of the subject that TeCePe has to cover on each moment.

Of course, not all units have the same relevance: some of them are more likely to make a bigger impact inside a sentence. For example, the sentence *"What is the difference between routing table and forwarding table?"* has two valid words for the context of a search (what and difference). However, it is obvious that the user prefers to obtain an element comparison rather than obtaining definitions. Therefore, words expressing comparisons should have a bigger relevance than adverbs. This should not be a problem as long as it has a correctly ordered structure.

Conclusions

In conclusion, by running both algorithms we can obtain enough grammatical elements to obtain user's concepts and intention to make a query to the database where the information is stored. In our case, these two elements are stored in a dictionary, composed of two fields:

1. "Concept": This key holds the name of the nodes searched and returned by the priority keyword algorithm.
2. "Context": This key holds the word that provides the context of the search.

This dictionary will be used as the information that will go inside the query to the database.

3.5. TeCePe external data format through database

Our system needs a database. Its role is storing information of possible answers that are outputted to the user. If this database is not modelled correctly, it might be difficult to have an efficient code and there is the risk of losing information in the process. Thus, it was decided to have two separated tables in a single MySQL database: one for the initial tree load and another one for storing answer's related information.

In order to store database's information, it was chosen that the model of the answers' table would have the following structure:

1. The "id_node" field is the node unique numerical identifier. Two nodes cannot have the same number. This will also be the key_search value of the MySQL table.
2. The "name_node" field stores the text of the concept/idea that users will search for. It is important to notice that words that require previous fathers have attached the most important father at the end of the word (for example, forwarding table entry will be tableforwarding) to differentiate equal words.

3. The fields “d_what, “d_how”, “d_when”, d_where” contain the answer based on the sentence’s context.
4. The “diff_node” field stores the value of the node’s name that a comparison can be done with (if any).
5. The “diff_text” stores the answer of the comparison.

The initial tree is stored inside the database, and it is loaded by TeCePe. This is very flexible, because the data manager can change answer’s information without modifying the code. Therefore, TeCePe will query this table and create each node of the tree through the use of the retrieved data, which have the following fields:

1. The “parent” field provides the parent node.
2. The “name_node” field provides the node’s name.
3. The “related” field provides the node’s name that must be present when performing a search. It can be NULL if no node is required.

Creating answers follows a more complex structure though. Therefore, how TeCePe obtains that information will be explained in later section of the document (Chapter 4.3.2).

The result of this process will always be a string of text, which will be the answer that users will receive, either being the search asked for or a failure message produced by introducing incorrect information.

3.6. Text correction

As our bot manages input text by users, it is very likely that it will contain errors. Therefore, a strategy must be chosen in order to deal with those mistakes. Two alternative options could be applied when the bot detects errors inside the text: trying to correct the mistake or informing the user about it. In this case it is more useful to tell the user what error might have been committed and provide possible alternatives.

In consequence, TeCePe will first analyse the text in order to check if there could have been any possible mistakes for the concepts of the tree exclusively. In other words, the rest of the text will not be analysed in this part of the process, as it is not relevant for detecting possible search mistakes. If found, then the search process will not be done. Instead, the bot tries to correct them by suggesting possible concepts that could have been the user’s intended one.

For example, if user’s input was the following:

What is iPe?

Before making the query to the database, the text is analysed in order to check possible mistakes. In this case, iPe is an error, so the search will not be done. Therefore, TeCePe tries to correct the error and reply with the possible(s) correct answer(s) that the user tried to search, which will be “IP” in this particular case.

3.7. Conclusions

TeCePe is a chatbot that serves as a portable encyclopaedia for users that require searching concepts about the protocols TCP and IP quickly and efficiently. There will be a clear separation between the interface of the bot and the backplane of the solution. It has some solutions to enable the best implementation possible: concept analysis, through the use of a mixture between the keyword searching and priority keyword searching algorithms: database searches and text correction.

4. IMPLEMENTATION OF THE SOLUTION

4.1. Introduction

Now that we have settled what a bot is and its main characteristics, it has come the time to talk about programming it.

First of all, we had to choose the most appropriate platform to host our bot. As we previously stated, not all platforms are the same in terms of content and presentation. Not only that, but also the tools that could potentially be used for its design vary depending on where the bot is developed. Therefore, all these considerations were analysed in order to make the right choice, as this step is a critical factor that affects the final product quality.

Our first option could have taken a more "original" approach, creating the bot from scratch or with, simply, the support of a mobile development tool like android studio. In many applications this is the best way to go, as developers are not constrained by external platform restrictions. Nevertheless, this approach has some drawbacks:

1. **Lack of a defined front-end:** Without the support of a platform, development process would have required not only programming the whole back-end, but also creating an *attractive* front-end. Taking into account this application is developed with time constrains, adding a good front-end modelling would certainly have taken time from back-end's development and, subsequently, the quality of bot's "heart" would decrease. Furthermore, designing an attractive front-end is not simple; applications like this one require an elegant, yet simple, presentation. Therefore, not having much experience with front-end development, programming it could have proven to be challenging and, subsequently, we could have ended up not getting a better result than a professional application can provide.
2. **Protocol management:** Besides a front-end, we need another programming layer in this approach: the connection between the bot and users. External platforms usually provide certain APIs that simplify workload and handle all connectivity issues that these platforms may have, which helps getting a better result and reduce development time.
3. **Popularity:** Many applications are uploaded daily in mobile platforms like Apple Store and Google Play. Chances for an application to become popular are quite low due to the big competition inside the mobile market. However, should the bot be hosted inside a familiar application for users, its accessibility would be higher and thus be easily spread among other potential users.

One important factor to be mentioned is that bots are a main focus for some of the most popular platforms, so they also provide certain tools that can make the coding of the bot far easier and less time-consuming.

4.2. Selected Platform

Having into account everything described in the above paragraphs, designing a bot without third-party support can be a very demanding and time consuming task. Therefore, it was decided to make use of one. In any case, this method can be a good approach even without having time constraints.

We want a relatively well-known platform that provides a good background for bot development. Two of them rise above the rest: Telegram and Twitter. They both satisfy our conditions: the two of them provide a friendly front-end, have a lot of popularity among users and possess certain useful APIs for bot programming. In this case however, only one platform can be chosen, so more facts were taken into account to choose the platform.

Although the popularity of Twitter is arguably bigger than Telegram's one, the total amount of users that each platform has is not relevant at those magnitudes. In our case, the factor that makes a significant difference is how much information each platform can display at a time in addition to its presentation.

Messages on Twitter have a specific format that demands using only 140 (280 depending on the version) characters per message. This prioritizes information synthesis to avoid overflowing users with non-relevant information. Although it can seem trivial at first, it must be taken into account that our application must provide definitions, comparisons… so the maximum length of a Twitter message can prove to be insufficient for certain contexts. Thus, information would need to be really summarised, which can derive into relevant information loss. Otherwise, it would need to be presented in different consecutive messages, which can be seen as an inconvenience to some users given the platform characteristics.

Telegram does not have the previous problem, as all its messages are displayed through plain text responses without noticeable character restrictions. Moreover, its interface allows the bot to simulate a conversation inside user's feed. Therefore, at least at some level, the bot might not seem as artificial as it would appear in Twitter, encouraging users to talk with the bot and experimenting with it.

Hence, with all previous considerations, we can safely assume that Telegram is the perfect choice for a platform to develop the bot, as it provides an environment with useful tools, an appealing front-end that encourages user-bot interactions, and has very high popularity among users all over the world.

4.3. TeCePe's general overview

In previous sections we defined all components required for our bot's correct functioning. In practice however, these components require not only their full development, but also a determined connectivity that is more complex than it might seem at first. Thus, to make bot's programming easier, those functionalities were divided in different programming modules that shared certain relationships. To visualize more clearly the modules' general connection, the structured chart on *Fig 4.1* was created:

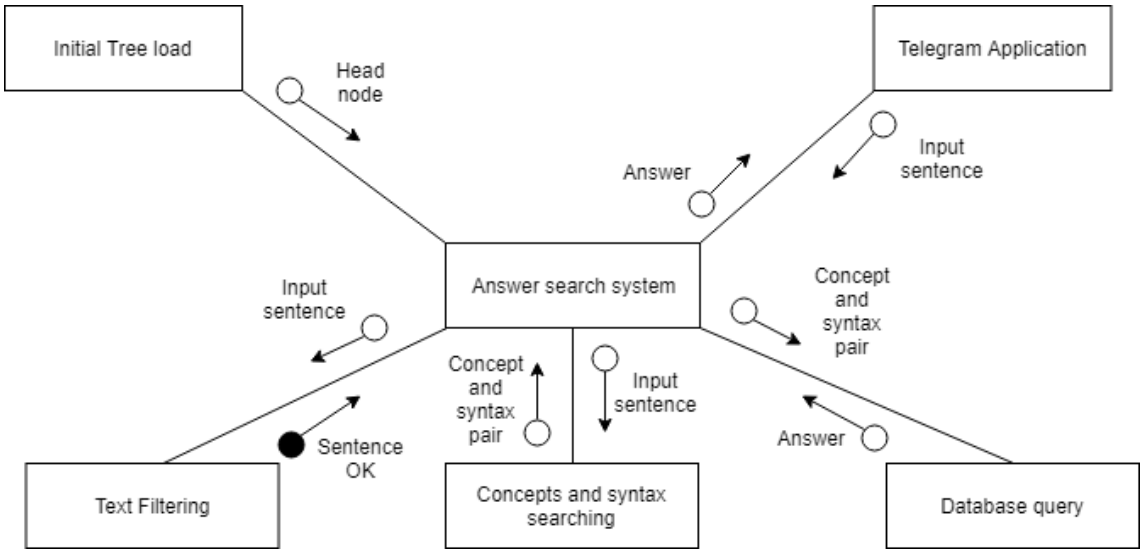

Fig. 4.1 TeCePe's structure chart

Each module satisfies one core functionality of the whole system. All modules possess certain structures that, due to its moderate complexity, will be detailed in the following pages. The only exception is the text filtering module, which is simply a run of the keyword search algorithm (explained in section 3.4.2) to locate bad words inside the sentence.

Initial Tree load

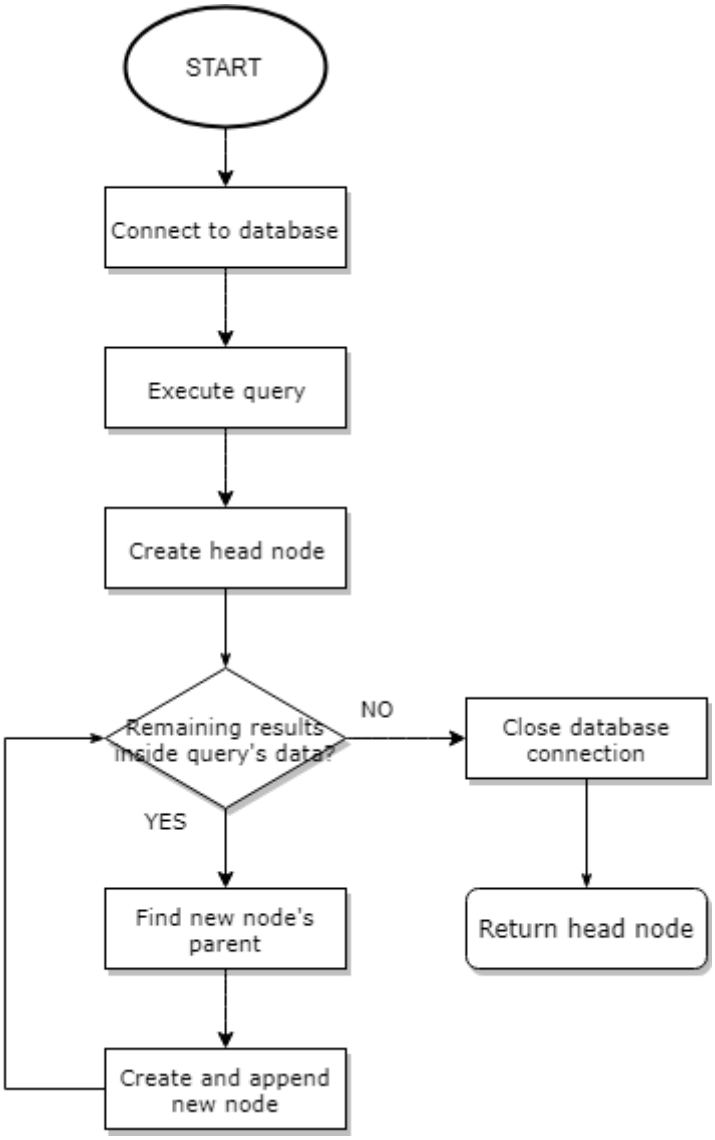

Fig. 4.2 Tree load flow chart

This module is the first step done by the bot, as all its general information must be loaded inside the system. In *Fig. 4.2*, as the initial tree information is stored in a database, it is necessary to retrieve that information through a query, iterating through all retrieved results' elements to define and create the nodes of the concept's tree. As defined before, the only node that is not present inside the database is the initial one. In consequence, that node is manually created in bot's code using the system's general node (usually the main subject). This module is only called once, specifically each time the bot is initialized.

Answer search system

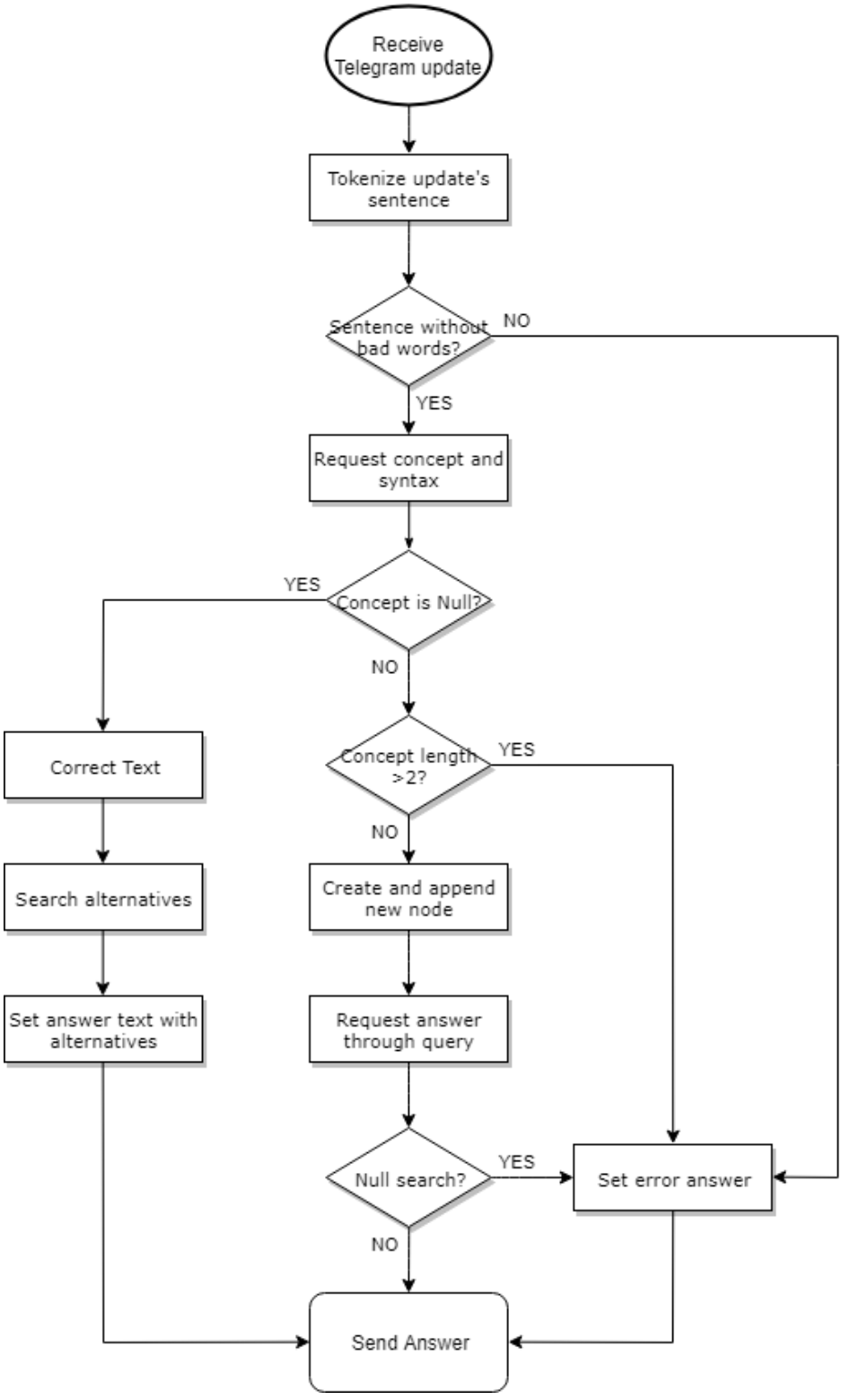

Fig. 4.3 Answer System Flowchart

This module is bot's controller: it acts as the link between the rest of modules and Telegram's interface. It receives updates from Telegram and processes them to output the right answer to each user depending on its input. To do so, the module tokenizes the sentence (as explained in section 4.9) and searches if the text contains any bad words by calling the text filtering module. If it does, an error message will be sent. Otherwise, the controller calls its "Concept and syntax searching" module to retrieve all sentence's concepts and syntax. If there are no concepts, then the sentence is corrected and all possible user alternative words will be sent to the user (if any). Otherwise, "Database query" module will be called to perform database's query to create the answer sent to the user.

Concepts and syntax searching

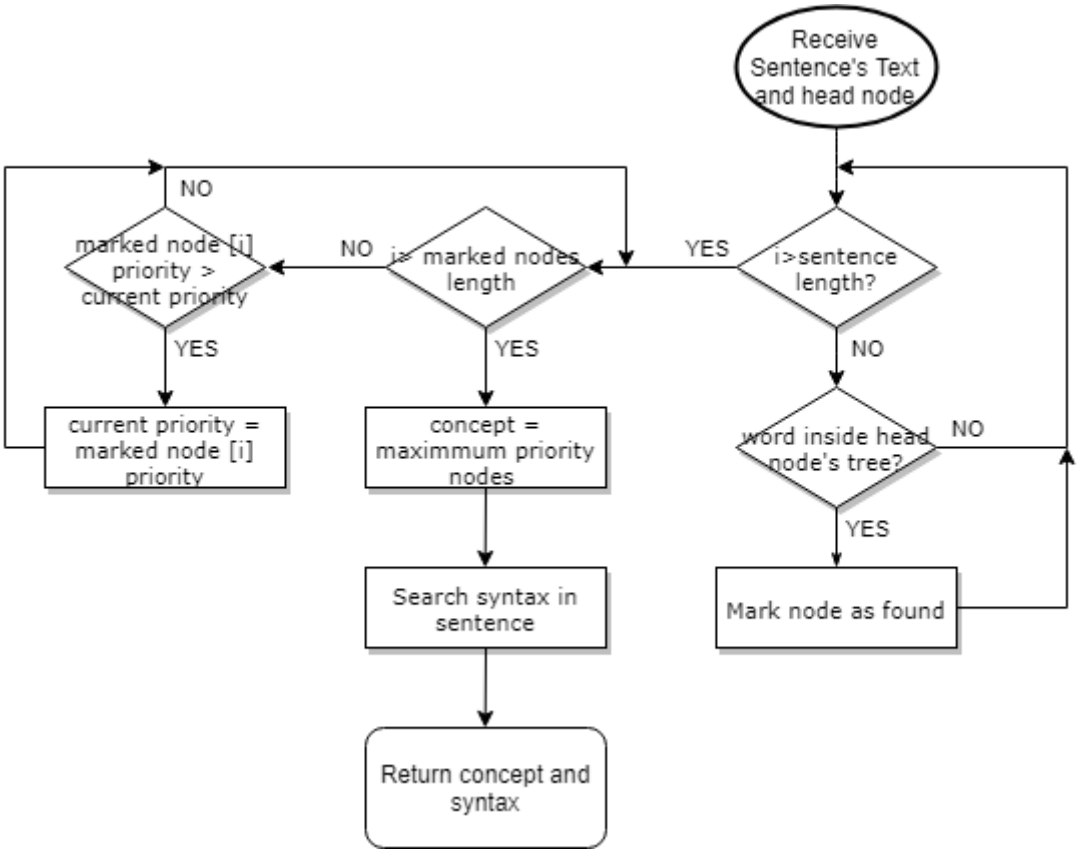

Fig. 4.4 Concepts and syntax searching flow chart

This module is responsible for checking inside sentences all ideas and context required to make a query to the database (algorithm explained in section 3.4.2). To do so, it iterates through all words inside a sentence, checking if they are present inside the tree. Afterwards, it compares priorities among nodes and takes only those with the highest priority value. Finally, it runs a simple keyword search in order to find the syntax unit.

Database query

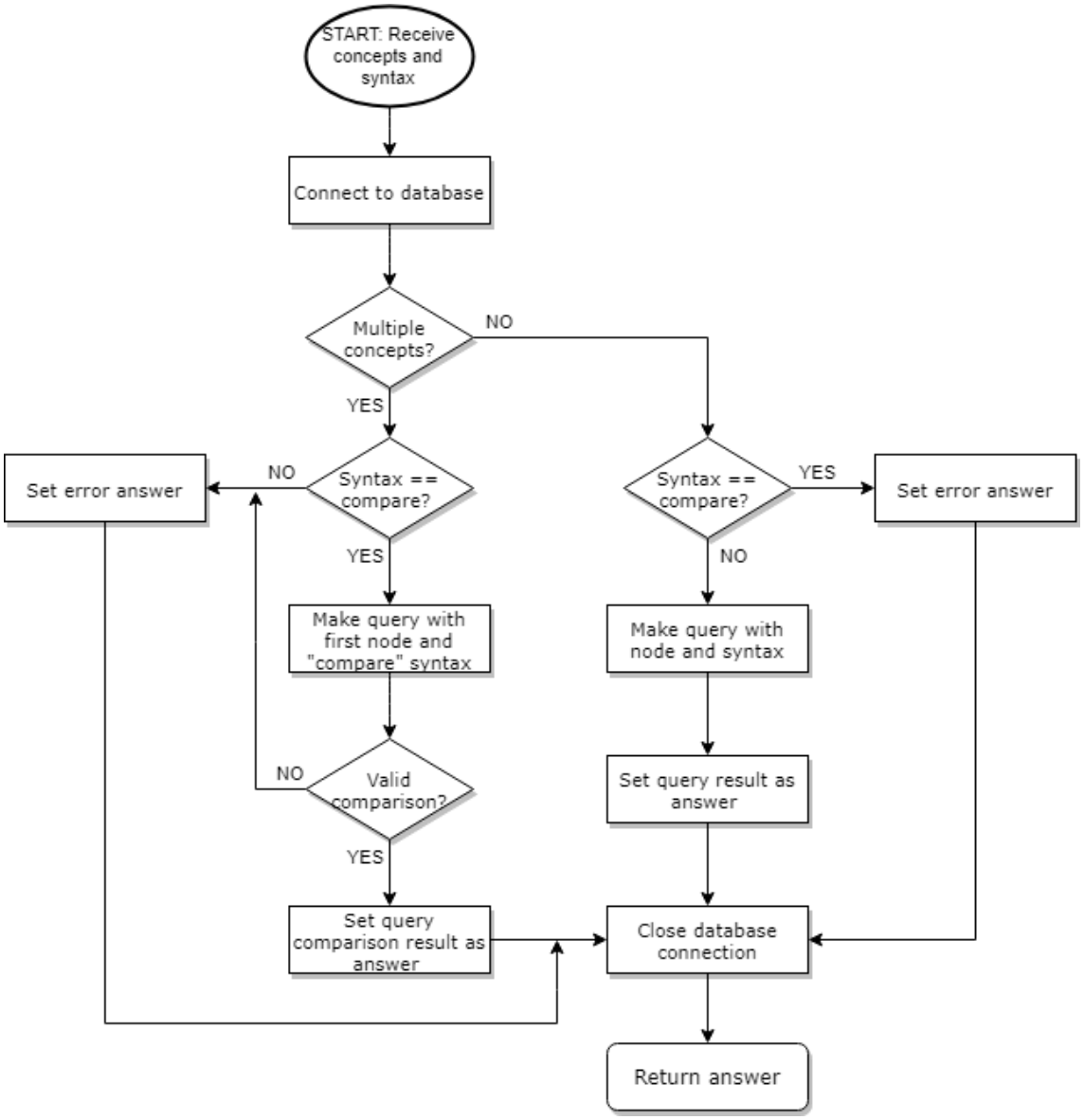

Fig. 4.5 Database query flow chart

This module is responsible for performing the query that will retrieve the final answer's value. It has two routes depending on the amount of concepts introduced. In both cases the module checks if the syntax has a comparative value, as it has special conditions regarding the number of elements present and requires checking if the comparison can be performed. If the amount of nodes found is more than one, then this check is done through querying the first node's "diff_value" column inside the database table (explained in section 3.5) and checking if both names coincide. If so, then "diff_text" will be set as the output of the module. Otherwise, an error answer will be sent.

In case that only one concept is retrieved, having comparison syntax will throw an error. Otherwise, a query using the concept node and its syntax will be done to retrieve the answer.

4.4. Creating a bot in Telegram

Telegram uses one bot to manage bot creations inside the platform: the BotFather (Telegram, n.d.). Its behaviour is pretty basic: when a user wants to create his own personal bot, he only needs to search BotFather inside the platform as he would do with any other user/bot. Once found, the command *'/newbot'* will ask BotFather to create the new bot. Therefore, it requires both a name to be displayed for every user and a surname for the bot. Afterwards, BotFather will generate an alphanumeric character string called the token. This is the key that allows the bot to use the Telegram bot API, necessary for managing connections between Telegram's s interface and the bot (Telegram, 2018).

BotFather is able to perform many other tasks. Some of them may include modifying bot's metadata, bot's deletion… In this case, two of those tasks were convenient for TeCePe: token and status alert generation. The former one is useful for "migrating" the bot from one token to another (this could be seen as creating a new session), while the latter one provides information about requests and activity reports useful during testing phases. Both will be further discussed in subsequent parts of this document (Telegram, 2018).

As we see, creating a bot in Telegram is a very simple process, especially with a platform providing powerful (and intuitive) tools to create bots, as well as managing them. However, this feature might pale in comparison with the API that telegram uses for bot's communication: the Telegram Bot API.

4.5. Managing front-end back-end communication: Telegram Bot API

As our bot is hosted inside an external platform, designing an interface-bot communication is mandatory. Furthermore, this management must be efficient, otherwise it could turn out to be sluggish and/or faulty, which would affect user experience negatively.

Hence, it is encouraged to use Telegram's API (Telegram, 2018) for this purpose, because its functioning is fairly simple.

First of all, the previously generated token provides authorization to use the API. Each token is unique for each bot's instance. However, more tokens can be generated through botFather if needed. This token will be used in the HTTPS queries.

Queries inside telegram are HTTPS and use the following format: https://api.telegram.org/bot<token>/METHOD_NAME, where <token> is the bot's

token and METHOD_NAME is the invoked method. Telegram allows using either the GET or POST HTTP methods.

The answer is a JSON object, which will always contain a Boolean field ‘ok’ with an optional ‘description’ String. As expected, if ’ok’ equals true, then the query was successful. Otherwise, the query failed and the error will be detailed inside the ‘description’ field.

The reception of Telegram updates uses two different (and mutually exclusive) methods: Updates and Webhooks. A webhook is basically an HTTP callback triggered by an event that will POST data to a URL for an application (Vegesna, Jain, & Porwal, 2018). In our case, webhooks are not useful as we are not interested in sending other data to different applications. Therefore, we are only interested in using updates, which are JSON objects with relevant information of the conversation between users and the bot. A variety of fields can be found inside this JSON, including:

1. User: Object representing the update’s user. Some of its fields can be id, username, is_bot…
2. Chat: This object represents the chat instance with the bot. It has some fields like id, type…
3. Message: This object represents the update message(s). Some of its relevant fields can be message_id, text, reply_to_message…

Other fields can be found inside this JSON object. However, most of them are optional and do not appear inside updates unless they specifically belong to their correspondent type, for example with the case of documents, stickers, etc.… These types are not relevant for TeCePe, as it only processes text. Therefore, every update must have only text. It is convenient to point out that TeCePe ignores not text-based updates by default, so every update that is not plain text will not receive any response from the bot.

Different methods to manage these updates are available inside the API. They perform functions like editing parameters, sending messages, getting files, etc.…

It is important to keep in mind that everything described above is compatible with every programming language available. Therefore, the next logical step is choosing an appropriate programming language to develop the bot. This is a vital decision, because using an appropriate language can help to make coding easier as well as assuring that the bot will be developed with the best possible environment.

4.6. Programming the bot: Python

The language chosen should be able to perform everything needed without being overwhelmingly difficult and having an active user-base to help solving problems that are encountered during development.

Having into account all those factors, Python is the language that provides the most advantages of all available languages:

1. Its syntax is very simple, eliminating as much as possible all comprehension barriers that other programming languages have. However, being simple does not interfere with the amount of functionalities that Python has. Moreover, it possesses certain advantages that other languages cannot provide (independence of variable returning type, dictionaries…)
2. Current user-base of Python development has never been higher. Every day more people start discovering Python and experimenting with its intricacies, leading to an almost unlimited source of available information online.

Above else, Python has one of the biggest pools of libraries, much bigger than any other language. These libraries make some programming tasks far easier by simplifying more monotonous codes to allow focusing in developing more challenging parts. It is also worth mentioning that many of them are open-source, so they can be used in program development freely.

4.7. Protocol management: python-telegram-bot

Precisely, one of the main libraries used is a wrapper of all the Telegram Bot API: python-telegram-bot (Toledo, 2015). This open-source library provides an elegant and simple interface to perform all necessary actions that require the Telegram Bot API usage. The main objective of this library is making Telegram HTTPS functions programming very straightforward in order to focus on bot's back-end development, leaving all communication handling for this library.

In order to understand better how it works, we will describe now the main elements of the library:

1. Updater: This element "listens" to the Telegram platform and catches its updates, sending them to a dispatcher (described below). Used inside the bot, a token parameter must be passed as an argument in order to be initialized, linking this updater with the bot and providing authorizations

to the library for operating with Telegram's Bot API. It runs in a different thread to allow users simultaneously perform other tasks.

2. Dispatcher: This element is responsible of receiving updates from the updater and decides the action taken based on the received information. In order to know these actions, it needs to have certain handlers (described below) attached.
3. Handlers: This element is responsible for defining the action that will be performed for each one of the received updates. Depending on the type of update, we can divide them into two different classes of handlers:
4. Command Handler: Responsible of all '/ [word]' commands.
5. Message Handler: Responsible of all received messages.
6. More types could be added (even generic ones or others specifically created for an application), but our bot only needed these two.

Now that all components have been described, we can see an example to fully understand how it works in *Fig 4.6:*

```
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters
from textblob import TextBlob
import logging

bot_token = [redacted]

updater = Updater(token = bot_token)
dispatcher = updater.dispatcher
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)

started = 0
forbidden = [redacted]

def start(bot, update, args):
        global started


        bot.send_message(chat_id=update.message.chat_id, text="Hello: My name is TCP, I love Telematics, so don't hesitate
to ask me anything you want to know!!!!!")

start_handler = CommandHandler('start', start, pass_args=True)
dispatcher.add_handler(start_handler)

 updater.start_polling()
```

Fig. 4.6 Python-Telegram-Bot example code

The first thing that needs to be passed as argument for the updater is the unique generated token. This way, the updater will be able to catch all updates from Telegram sent to our bot.

Then, the dispatcher is stored inside a variable to pass the different handlers required for every bot's action. These are defined as a function that, when invoked by the dispatcher after some input, triggers up to execute a defined behaviour. In our example, TeCePe simply replies with some text to the '/start' command, as seen in the function "start (bot, update, args)". The details of each particular command are defined in the new 'Command Handler' generation, where details like words used for invoking each command or passing arguments are defined. Once a handler is set, it must be added to the dispatcher.

Finally, the updater has to listen for Telegram updates regarding bot's interactions. Therefore, the last line on the example code ("start_polling()") enables the updater to listen at all times possible updates, preventing the termination of the program.

Now, as it can be seen in *Fig 4.7,* the bot replies to the '/start' input with the expected message:

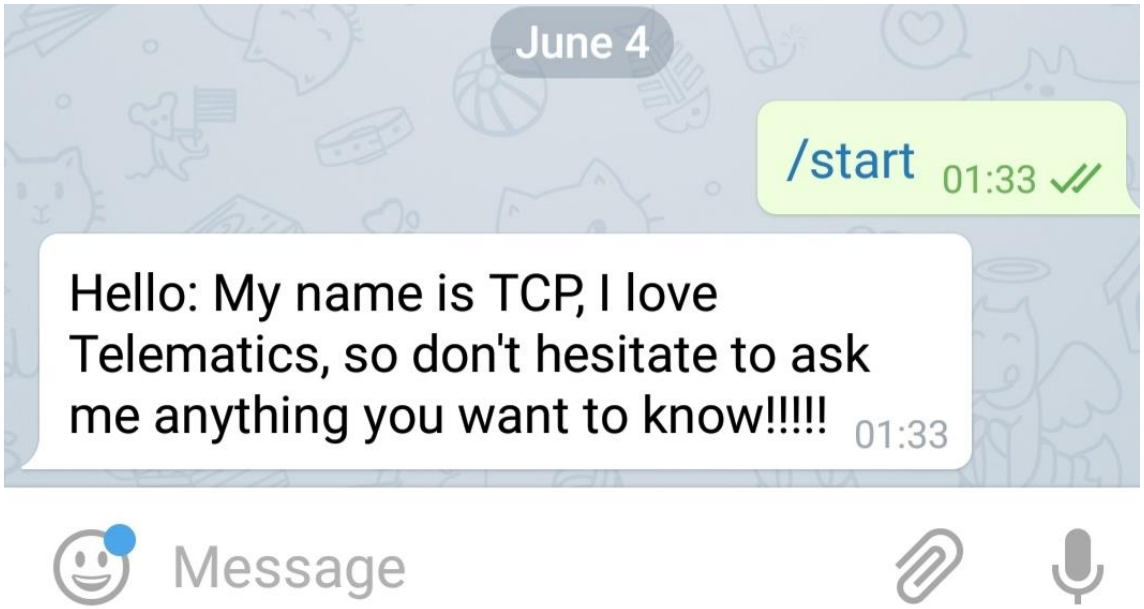

Fig. 4.7 Telegram response

In conclusion, all users-bot communications are handled by the updater and dispatcher, while the rest of the application is located inside the different handlers functions defined for every input/action received.

4.8. Telegram syntax and format

Based on bot's functionalities and desired behaviour, the different commands and actions considered for the application were the following:

1. /start: Many bots inside Telegram require this command in order to start its execution. It is very useful in applications that require a "soft-start" or games. In our case, this feature could be used to communicate some important information regarding the bot. First, putting a disclaimer to inform users that its information might be faulty and should never substitute teacher's explanations and to contrast with other sources. Also telling the user that all information used for the answers was obtained from Kurose's book (Kurose & Ross, 2012).
2. Plaint text input: mandatory for text processing done by the bot.

3. /close: Also represented in other applications as /bye, this command terminates the execution of the program or user's chat. In our case, it seems pointless if no "/start" command was developed.
4. /help: This command is often used to provide information regarding bot functionalities or manuals. In our case, this simple command is used to specify certain aspects of the bot and explaining how to interact with the bot.

Now that we have completed the description of everything regarding Telegram-bot connection and the different actions we have taken while creating the back-end of the bot, we can see the tools used to create one of the bot's main features: Natural Language Processing.

4.9. NLTK & TextBlob

So far, we are able to retrieve text from user's input. However, processing text without using proper mechanisms is a very challenging task, especially when it can grow into large volumes of data. In consequence, some sentence tokenization is required in order to create separated entities that can be independently manipulated with relative ease.

Human language is not easy to segregate into smaller units: the own nature of human languages has been studied for centuries and never reached perfection, evolving every time more and more. Hence, translating this analysis into machine processing is a very difficult task for its complexity and requirement of certain analysis like ambiguity, sentiment and speech …

Fortunately, most tokenizing (and analysis) work has already been done in Python. NTLK (acronym for Natural Language Toolkit) is a Python library created for the purpose of helping with natural language analysis, providing developers with a large set of tools to work with natural language data while also having functions for tokenization, parsing, etc… (NLTK Project, 2018).

Firstly, it is important to define what parsing actually is. Parsing is the ability to separate a sentence into grammatical parts, such as subject, verb, etc. Moreover, in machine language processing is the ability to perform these separations to create some structure that can be analysed to check the grammatical and syntactical correctness of a sentence.

Thus, it is required to effectively separate strings into independent elements. At first glance, a separation based on white spaces and basic punctuation symbols could be a way to solve this problem without relying on external libraries. However, this approach is too basic, as natural language is far more complex. It has some flaws: defining valid punctuation symbols, white space location… Furthermore, programming this

functionality could be very resource consuming due to a possible poor implementation, making the program slow and unreliable.

NTLK avoids this problem by providing a tokenization function. Basically, the library allows tokenizing sentences in words and punctuation signs universally accepted in the English language (or any other language selected), returning a structure of separated elements to create a more manageable structure.

Although tokenizing is very useful for natural language data processing, it is necessary to locate both grammatical and syntactical element types, not only its fragmentation. For this purpose, NTLK can get some string of natural language data to create a structure that shows all construction details of a sentence, like the type of words used, grammatical unit... The most used representation are trees, as they usually provide the best visual information compared to other structures like tables. For example, we can see how NTLK would parse the sentence "Pierre Vinken, 61 years old, will join the board as a nonexecutive director" in *Fig 4.8:*

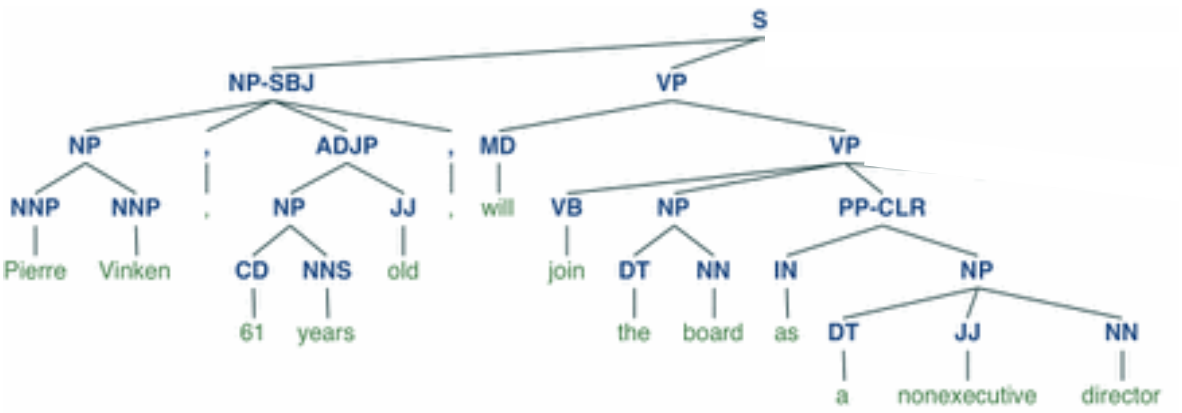

Fig. 4.8 Parsing Tree (NLTK Project, 2018)

NTLK creates the tree separating each word and punctuation signs into different syntactic elements. Each level of separation becomes more specific, locating the word and its role inside the whole sentence. This is very similar to how traditional syntactic analysis works.

Despite all benefits that the NTLK library provides, it is not perfect. One of its main flaws is the way NTLK manages its data. Each one of their functions has a completely different output structure. For example, tokenization outputs a string, but tagging outputs a double string using the previous tokenization, so the previous variable has to be generated even though it might not be necessary. Therefore, we can end up in situations where we might need more variables stored at the same time for different purposes, increasing the complexity of the code, or having to perform the same operations at all times, reducing the efficiency of the code. This example in *Fig 4.9* will help to visualize this situation better:

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

Fig. 4.9 Some functions of NTLK (NLTK Project, 2018)

It is clear that, if we want to keep both functionalities available, we should adjust certain parameters inside the code. Two main solutions could be done in this case: we can operate with the initial string at all times, performing the necessary transformations each time they are required, or we can operate by passing all entities as parameters when needed, which would increase the complexity and keep more variables occupied, reducing efficiency. Hence, neither of the solutions would be beneficial. Therefore, we needed a unit that would allow storing all that information while keeping their functionalities intact.

TextBlob is an open-source Python library that provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more (Loria, 2018). It is very simple and easy to use when compared to the NTLK library. It takes the most important NTLK processes and simplifies their presentation, allowing programmers to use a simpler interface while solving some minor issues of the NTLK library structure. Every element is a TextBlob unit, created from a sentence (string) of natural language. It has some downsides however, as it does not provide the whole set of NTLK functionalities. Nevertheless, TextBlob has almost everything required for TeCePe's programming.

How TextBlob works can be seen in the small program in *Fig. 4.10* that showcases some library behaviours and its implementations in Python:

```
from textblob import TextBlob
from nltk.stem import WordNetLemmatizer

question = TextBlob("IP protocol is cool, but TCP is cooler").lower()
question_es = TextBlob("Hola soy TCP, encantada!!")

def find_noun(sentence):
    for word, word_type in sentence.tags:
        if word_type == 'NN':
            if word == 'ip':
                answer = "IP is a protocol of the Network Layer"
            elif word == 'netmask':
                answer = "The mask is used for subnetting"

    return answer

def locate_word(sentence):
    if 'tcp' in sentence.words:
        print 'That is a great protocol indeed'
    return sentence

print question
print question.words
print question.tags
print question.parse()
print find_noun(question)
print locate_word(question)
```

Fig. 4.10 TextBlob example code

First of all, TextBlob creates the structure by simply passing a String as an argument. From now on, all operation should be done with this structure that, in some way, behaves like a regular string with added capabilities, even allowing string comparison and/or concatenation. For example, TextBlob content can be changed to lower case letters with the command *'.lower()*', or doing the opposite using *'upper()'*, like it would be done in any other Python string. In the previous code in *Fig 4.10*, the output of printing the question variable (the textBlob itself) would be:

"ip protocol is cool, but tcp is cooler"

Notice that output sentence would be exactly the same as if the executed command was printing the same string in lower case, illustrating that its behaviour is fairly similar to the normal string ones.

There is no limit on the language used inside TextBlob: it will recognize the language, if possible, using the command '.*detect_language()*' . It can even try to translate it into another language with '.*tranlate(from=[language_input], to=[language_output])'*, though the best quality of the translation is not guaranteed.

The command *'.words'* will output a Python WordList with all the sentence words, performing the desired tokenization. This WordList structure is a Python list that has

some additional methods, so it can be treated as a traditional Python list. For example, the output of the *'print question.words'* command would be:

['ip', 'protocol', 'is', 'cool', 'but', 'tcp', 'is' cooler']

The list has exactly the same structure as a Python list with the correct tokenization.

Another example of this behaviour can be found in the '*locate_word(noun)'* function, which locates a particular word with the instruction 'in' for the words inside a textBlob wordlist (as we would normally do with a normal list). This function (*locate_word(noun))* would output the following:

"That is a great protocol indeed"

It located successfully the requested word among the textBlob ones, so the function returns the correspondent message.

Furthermore, textBlob allows performing more advanced tokenization methods through the use of a sentence tokenizator, which are defined inside the NTLK API. However, TextBlob makes its coding far more direct and simpler without losing this NTLK's functionality.

TextBlob supports text parsing with the *'.parse()'* command. In our example, similarly to the NTLK one, the output is:

ip/NN/B-NP/O protocol/NN/I-NP/O is/VBZ/B-VP/O cool/JJ/B-ADJP/O ,/,/O/O but/CC/O/O tcp/NN/B-NP/O is/VBZ/B-VP/O cooler/JJR/B-ADJP/O

As can be seen, the output has the tree shape expected in plain text. In order to represent it graphically, some additional code would be required.

TextBlob also allows creating a mixture between tokenization and parsing with the *'.tags()'* command. This will create a 'part-of-speech tagging' Python list, showing all sentence words with their correspondent element tag, indicating their function inside the sentence with NLTK syntax:

[('ip', u'NN'), ('protocol', u'NN'), ('is', u'VBZ'), ('cool', u'JJ'), ('but', u'CC'), ('tcp', u'NN'), ('is', u'VBZ'), ('cooler', u'NN')]

Here, all list elements are composed of a pair of values: the word and its grammatical unit in NTLK syntax. This is useful to solve ambiguities in some sentences. For example, both sentences "*I want to know the route to 128.2.3.4*" and "*We need to route the packet to this other network*" have the word route inside. However, their context and

meaning are completely different in both cases, being the first one a verb and the second one a noun, so it is important to realise which context is used at each time.

All functions described are just a scratch of all functionalities textBlob can offer. However, only those required to create the back-end of TeCePe were used, so there is no need to describe others in more detail inside this document.

4.10. Anytree

Python does not have a default tree structure. Usually, it is normal to define the tree class that suits better for the purpose of each project individually, as it is not very difficult to program and design it. However, in our case, we use one Python library that greatly simplifies this task: Anytree.

Anytree is an open-source Python library that provides a simple interface to work with tree structures. The basic tree unit is the Node, which also has different subclasses with different characteristics. In our case, we used the Node structure, which is able to store a name and any other arguments passed as parameters.

This library performs basic tree-oriented functions like node searching, depth calculation, node attachment and detachment... Furthermore, it also provides more advanced functionalities such as tree loading based on a dictionary or a JSON file, as well as exporting to those formats.

In order to check how it works, we have a simple example in *Fig 4.11* to showcase its main functionalities and advantages over classic tree programming approaches:

```
from anytree import Node, RenderTree
from anytree.search import find, findall

head = Node("ip")
son_1 = Node("routing", parent = head)
son_2 = Node("forwarding", parent = head, added = "More info")
grandson_1 = Node("table", parent = son_1)
grandson_2 = Node("table", parent = son_2)

print head.name
print RenderTree(head)
print findall(head, lambda node: node.name == son_1.name)
print findall(head, lambda node: node.name == grandson_1.name)
print grandson_1.depth

newson = Node("addressing")
newson.parent = head

print RenderTree(head)
```

Fig. 4.11 Anytree example code

The Node object is initialized by simply passing a string, which is the name of the node. If a node is a son of another one, then the parent argument must be defined inside its declaration. In order to access the tree at any point, the initial node "defines" the tree. Therefore, any function that uses the tree will need this head node to operate with the whole tree.

To access any node attributes, it is necessary a simple reference to it, as can be seen in the first print function of the script (print head.name), whose output is "ip".

AnyTree has some defined functions for searching elements inside trees. The function *"findall"* returns all elements with a certain name. It requires the name and the head of the tree. The use of alternative functions, like "*find*", is discouraged, as it is not able to search more than one node. If more nodes with the same name are found using this function, then an exception will be thrown. Besides, "*findall"* can also find single nodes without any problems.

The "*findall*" output of is a list of nodes with the correspondent name and path inside the tree. In our case, for the first *findall* present inside the code, only one node will be outputted:

(Node('/ip/routing'),)

In the second instance, the output is:

(Node('/ip/routing/table'), Node('/ip/forwarding/table'))

Anytree also has rendering capabilities, as it can draw the shape of any tree that uses the library. In order to do so, it is necessary to use the "*RenderTree*" function, which requires the tree's head node. This function also accepts multiple codifications (like ASCII) with the style parameter.

In our example, in the first RenderTree(head) in the code in *Fig 4.12* would output the following:

```
Node('/ip')
├── Node('/ip/routing')
│   └── Node('/ip/routing/table')
└── Node('/ip/forwarding')
    └── Node('/ip/forwarding/table')
```

Fig. 4.12 RenderTree first output

Drawing the tree from the previous output is pretty straightforward:

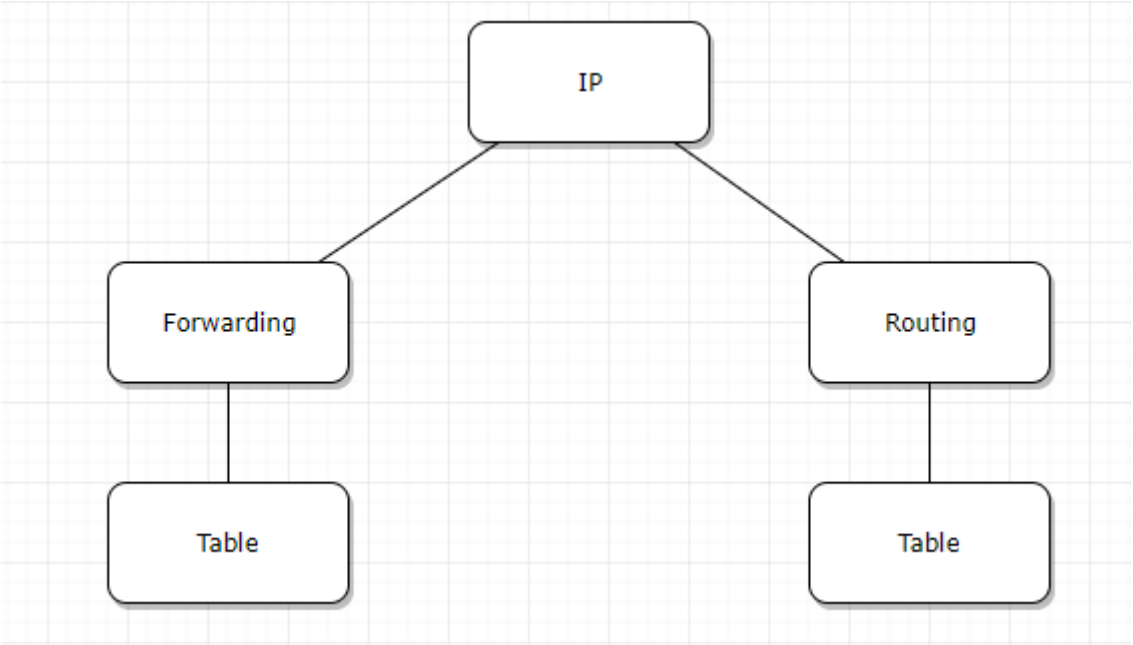

Fig. 4.13 RenderTree output graph

If needed, adding (deleting) nodes requires only the analogous attach (detach) functions from other languages. In *Fig 4.14* and *Fig 4.15*, the rendered tree after node's attachment would be:

```
Node('/ip')
├── Node('/ip/routing')
│   └── Node('/ip/routing/table')
├── Node('/ip/forwarding')
│   └── Node('/ip/forwarding/table')
└── Node('/ip/addressing')
```

Fig. 4.14 RenderTree output after adding the addressing node

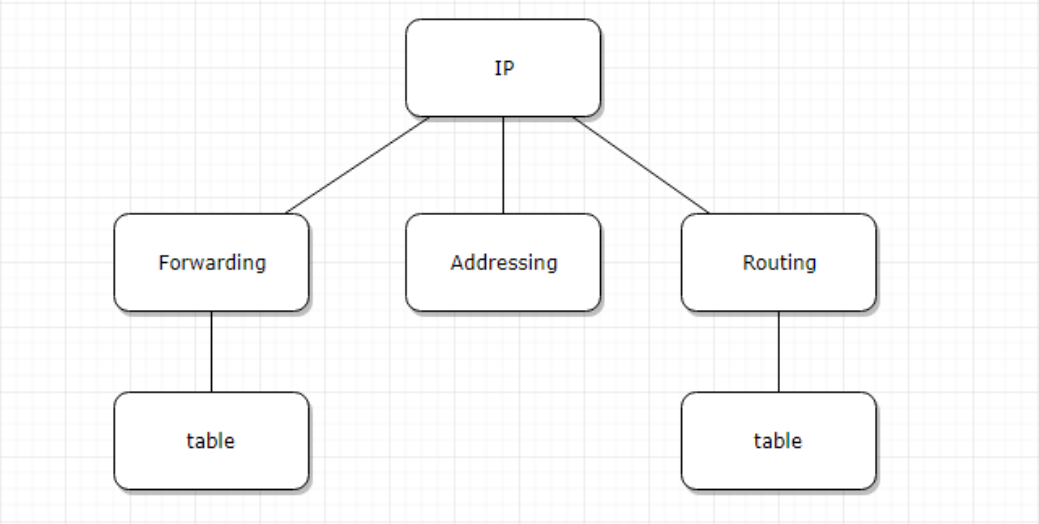

Fig. 4.15 Addressing node inserted inside the tree

4.11. PyEnchant

Having into account that TeCePe incorporates some text processing and analytics, the bot has to somehow be protected against user's faulty inputs. Although it is usually user's own fault, not managing correctly these situations can lead to a bad perception of the bot, lowering its good reputation among users.

Using text detection and correction mechanisms is not as straightforward as it can appear at first glance. Many things have to be taken into account (word distance, correct parsing, etc.). Moreover, the solution has to follow certain additional requirements that our bot introduces:

1. High accuracy: It is essential that almost none of the important search elements are lost or misinterpreted during correction. The main reason is clear: our bot is very sensitive to the names appearing in the nodes inside the tree. If the input is slightly different from the nodes content, then the search will fail or will be incorrect. Thus, the implemented solution must have the capacity of correcting as many nodes related words as possible. For the rest of the sentence is not required, though desired, as it is not fully relevant to the bot's internal functioning.
2. The proposed solution must be English based and, if possible, easy to be changed into other languages. The solution must have this capacity because English is the bot's language. However, it should also be possible to have an easy switching to different languages in the hypothetical case of expanding TeCePe's reach.
3. It must allow introducing a set of personalized words as part of its dictionary: The main problem with many words managed by TeCePe is that they are not part of the English language, either because they are too

specialized or are word acronyms. Hence, they are not usually included in dictionaries, so they have to be introduced manually or, in the worst case, have to be managed externally while keeping efficiency as high as possible.

TextBlob incorporates an error correction tool. However, it is not recommended for applications like TeCePe, as it fails in many aspects that we require: its accuracy is only around 70% for English language, and it is even lower for other supported ones. Moreover, it does not allow introducing personalized word lists.

Fortunately, we can take advantage of the Python's library friendly environment in order to use another package for our correction process: PyEnchant (Kelly, 2018).

This Python library is an implementation of the Enchant API, used to check word spelling and get correction suggestions. It can use many popular spellchecking packages including Ispell, Aspell and MySpell. It is quite flexible at handling multiple dictionaries and languages (Kelly, 2018).

With this API, we satisfy all previous conditions discussed before: multiple languages are supported (being English and all its variants the main one of all), library accuracy is pretty high (estimations suggest it could be around 90%) and it allows introducing personalized word lists. The only library's downside is that is no longer supported by its developer, so if any bugs are present, it is unlikely that they will get eventually fixed, so an alternative solution should have to be developed instead.

We will see how this library works by using a simple example shown in *Fig 4.16*:

```
from enchant import DictWithPWL
from enchant.checker import SpellChecker

my_dict = DictWithPWL("en_UK", "mywords.txt")
my_checker = SpellChecker(my_dict)

my_checker.set_text("This iPe is addresses with ip-v6.")

custom_words = []

with open('mywords.txt','r') as f:
    for line in f:
        for word in line.split():
           custom_words.append(word)

print custom_words

for error in my_checker:
    print "ERROR:", error.word
    replace = error.suggest()
    print replace
    answer = set(replace) & set(custom_words)
    if len(answer) == 0:
        error.replace(replace[0])
    else:
        error.replace(next(iter(answer)))

print my_checker.get_text()
```

Fig. 4.16 Pyenchant example code

It is mandatory to load a dictionary from a wide variety of options available (in this case, British English). The dictionary allows introducing personalized words from a .txt file. This text file must include all the words separated by line spaces.

The spell checker must be initialized with the previously created dictionary. This unit detects errors and performs corrections based on the selected dictionary. To introduce some text we used the command "*.set_text"*. In our case, the sentence chosen is "*This iPe has addres 192.168.0.0",* which has two errors: "iPe" and "addres".

To check which errors were detected, a "for" loop is required to iterate through all of them. PyEnchant will select a series of replacement words based on the dictionary for every error (in this case, British English and our personalized words). The command "*.replace*" will change the error for a new word. The common behaviour is usually replacing the error with the first word from suggestions, as these ones are based on parameters like word distance, likeliness, etc.… taken from spelling error analysis algorithms.

Unfortunately the checker does not allow modifying priorities among dictionary's words, so we could not prioritize our personal words list over the rest and we did not know how to find where those words were inside the replacement list. Hence, it was mandatory to find an alternative solution.

Before making a correction, the list of words from the text file is stored inside an array with the use of the *open* Python command in order to "load" the words inside the script. With both the replacement list and the custom words list, for every error inside the checker, a set of words is created with the common elements between them.

In the example case, the word "iPe" can potentially be inside the replacement set. However, the errors obtained inside the sentence show it is not the first option, so using the custom_words list we can obtain the set that has "ip" as correction on its first position and, in consequence, will be chosen as the replacement. The same case happens for the second error.

After the process, the output of the checker can be obtained using the function "*get_text()"*, which returns the content as a string. In our case, the output will be:

The ip has address 192.168.0.0

4.12. MySQLDB

As TeCePe manages large amounts of data with the responses, it is not a good approach to store that information locally; not only would it be far less accessible for modifications, but it would also affect negatively the efficiency of the code.

Instead, the most recommended course of action to store and retrieve data is using a database, where all information is gathered with a certain order. Therefore, we had to choose which database is better suited to our needs to perform this storage, as there are multiple valid choices (MongoDB, Solr…). In TeCePe's case, MySQL database is the best possible solution, as the information managed is neither especially delicate nor too extensive to require a more advanced one.

MySQL works using tables contained inside every created database. These tables hold their values using columns as table's fields and rows as its entries. When creating a table, the name of every column and datatype that itvbelongs to must be specified, as well as every other parameter that each value might require (Not NULL, INCREMENTING…). TeCePe does not create, modify or delete any of the tables; it only retrieves data from them.

In order to locate a value, it is necessary to make a query, which has this form:

SELECT [field] FROM [table] WHERE [condition (not mandatory)]

The information type that this query has depends on the table parameters, but usually has types like strings, integers… that would be found in other programming languages. The information is retrieved as a set of rows, one for every obtained value.

All previous command descriptions have been defined for direct database management, in other words, for having a connection through the use of the system's command prompt or "MYSQL Workbench". However, as TeCePe uses Python, this process cannot be manually done.

In order to operate with the database, the direct connection between the script and the database has to be established. Afterwards, the query must be done using that connection and data will need to be "transformed" to be operative. Python does not have the necessary tools to perform the connection. Therefore, it is mandatory to use an external library: MySQLdb.

MySQLdb is a Python library created by MySQL that allows Python applications to connect to any MySQL database and retrieve data and/or modify its content.

This library uses "*cursors*", a variable that points to a given database and "translates" Python syntax into MySQL and vice versa, transforming data from any query into a Python-readable form. *Fig 4.17* is a simple example for loading some nodes inside TeCePe to illustrate the behaviour of the library:

```
from anytree import Node
from anytree import RenderTree
from anytree.search import find, findall
import MySQLdb
import logging

db = MySQLdb.connect(host = "localhost", user="root", passwd = "admin", db = "TCPIP")
cursor = db.cursor(MySQLdb.cursors.DictCursor)

query = ("SELECT * FROM Tree_nodes_example")
cursor.execute(query)
found = cursor.fetchall()

head = Node("ip")
for row in found:
    parent = findall(head, lambda node: node.name == row['father'])
    #assuming database results are in order, the last element is the one that will be the parent
    newNode = Node(row['Node'], parent = parent[-1])
    print row['what']

cursor.close()
db.close()
print RenderTree(head)
```

Fig. 4.17 MySQL example code

The table from the database used is the one on *Table 4.1*:

Table 4.1 QUERY RESULTS

Id	NodeID (Primary Key)	Node	Father	what
1	1	routing	ip	Routing Definition
2	2	forwarding	ip	Forwarding Definition

The first step is establishing a connection with the database using the "*.connect"* command with all its required parameters: host (will vary depending on database location), user, password and objective database. This connection must be stored inside a variable to be accessible. In our example, we connected to the database TCPIP in localhost.

Subsequently, it is necessary to create the cursor to operate with the database tables. There are many types of cursors available, but the best one for our purposes is the "DictCursor", as it allows accessing rows' fields with the column name. After setting the cursor, queries should be done using the "*.execute*" function with the format described above (in this case, all rows from the table "*Tree_nodes_example*"). The "*.fetchall*" function retrieves all results in a row format.

To access all values from the search, a for loop must be used to iterate through all rows. All information can be accessed by simply using the table fields as array indexes. In this case, we access the routing and forwarding definition, as well as their names and fathers to build the tree. The output would be the following shown in *Fig 4.18*:

```
Node('/ip')
├── Node('/ip/Forwarding')
└── Node('/ip/Routing')
```

Fig. 4.18 Tree retrieved from the data base

As we can see on *Fig 4.18*, the tree has no differences with a hard-coded one. Hence, retrieving the tree from the database is more efficient and produces the same results as hardcoding the tree in the code does.

It is important to close both the cursor and the connection to avoid possible connection problems in future database callings.

4.13. Conclusions:

From all possible frameworks that support chatbot development, choosing Telegram to host the bot is the most sensible option that could be taken due to its bot-friendly environment and powerful connection API. In consequence, the most suitable programming language for bot's requirements is Python, because it allows the usage of very powerful libraries to develop all basic functionalities that TeCePe needs. These functionalities are separated in programming modules that when connected, will produce the desired behaviour.

We have shown an analysis of the technical solution of the bot. We performed tests to check the correct behaviour of all bot's components. However, the validation of the behaviour bot required tests with users. These tests were carried out in order to localize all possible bugs and flaws inside the system, to correct them and to obtain feedback to make improvements that were not considered during development. The validation work is described in the next chapter.

5. VERIFICATION TESTS

5.1. Introduction

In this chapter we are going to see how TeCePe was tested to verify bot's functioning and obtaining feedback from users. Thus, this process was divided in two different parts to have a better understanding of the application's behaviour, describing its deployment and reached conclusions through user's feedback.

5.2. Bot deployment: local verification

During development everything was done locally in the PC, including testing programming functions and checking general behaviour. The bot Python script had to be run for short time periods when necessary, transforming the PC the host of the bot, which would be impossible in a real scenario as it would need to be running 24h for complete availability. Therefore, it was mandatory to deploy TeCePe inside an external server to have a completely working application at all times.

For this task, it was required to transfer the bot from the local environment to the server by using an SSH connection. This was more than transferring TeCePe's script to the server: all external files and MySQL databases had to be migrated too. These lasts ones required further processing to make this possible.

A new MySQL database had to be created inside the server to store all PC's local tables, as well as setting a new MySQL connection with its parameters (defining both a user and its password). Afterwards, a new .sql file was created with all queries required to create and copy all tables from the old database into the new one. Executing this file with MySQL in the server would introduce the information inside the new database. Hence, we ended up with the same information in both databases. These steps were required every time a modification was done to the tables, as it is discouraged to modify tables in production environments, like the server.

The Python program had to be running in the background at all times in order to have the service available 24h. Unfortunately, the first run performed failed due to a connection problem. It was a simple problem to solve however, as the MySQL connection parameters were not adapted in the code. If those did not coincide with the previously configured ones, then MySQL would deny the connection. This prevented TeCePe from connecting to the database. Hence, changing those parameters solved the issue.

After assuring that TeCePe was permanently running in the server, more tests were required to check bot's behaviour before proceeding with the test with users.

These new rounds of tests were done through an EXCEL composed of certain amount of inputs that involve all TeCePe's modules and those ones that could pose a challenge to the connection. Each input has two columns:

1. Expected result: Expected behaviour from the bot.
2. Obtained result: Actual behaviour received.

Each successful test is marked in green, while failed ones are in red. These last ones also have another column detailing the possible reasons why the test failed and, if possible, provide a potential solution.

The main objective of these tests was locating potential connection problems that might have appeared during the migration process while also checking if the rest of TeCePe's functionalities were still working properly.

The general results proved that the migration process was successful. Therefore, no further changes had to be done in TeCePe's code after altering its database connection parameters. Obviously, the server's script is never used for any other purposes than testing and providing the service to its users. Therefore, all development was still done in our local environment; only after making all changes the code could be uploaded again.

5.3. Verification with users

Testing locally is a great way to find problems that may have arisen during development phases. However, these tests are not enough to provide definitive results about bot's quality, as they are usually carried out by the own developer, and do not provide feedback about design problems or possible improvements. Therefore, it was mandatory to design a user testing phase to ask user's for feedback about possible bugs, design problems and improvement that could be made.

These tests were done through a simple survey asking details about some general information regarding the experience of users with conversational bots and TeCePe. Because TeCePe could be accessed through Telegram, as long as users had access to that application, no more technical requisites were needed to participate in the survey. However, based on the subjects covered by the bot, it was recommended to possess some knowledge about the IPv4 protocol and general routing concepts.

The survey was distributed among telecommunication students of two different countries (Spain and Portugal) in order to reach as many users as possible. This survey was divided into three main sections:

1. My experience with bots: In this section, users were asked to grade from one to five their agreement with some statements. These statements covered some ideas

regarding bots perception from users. Being more concrete, these questions asked if users had any previous experiences with conversational bots, if they could be useful on their subjects, general preferences for bot behaviour and preferences about information display.
2. Fixed questions: To introduce users to how TeCePe works, they were asked to introduce some question and grade from one to five three main aspects of the experience: how fast the search was, to what degree the question was answered and how clear the definition was.
3. Free talking: After introducing users to TeCePe general functioning, they were asked to talk with it during an interval of five to ten minutes. Afterwards, they had to fill two questions, one regarding possible problems, bugs or issues they may have encountered during this phase, and the other one for providing feedback about improvements that could be implemented inside TeCePe.

12 students answered the survey. Although the number is a bit low, we can obtain a general idea of users' conclusions. The general perception of the bot was positive. Furthermore, the survey provided a useful set of conclusions that can help improving TeCePe in the future. Concretely, the main reached conclusions have been the following:

1. Users prefer having bots inside popular applications instead of being a separate entity.
2. TeCePe's search speed is highly appreciated among users, which generally think it is very fast replying queries.
3. Users generally find answers correct. However, as loaded data was still very rough during this survey, the content of the answers was not as highly regarded, asking for better and clearer definitions that would provide better quality information.
4. Users generally have limited experiences with chatbots. Although most of them realise some of TeCePe's potential to some extent, others seem slightly dubitative about uses of bots like TeCePe and how they could help studying telecommunication subjects.
5. Regarding information display, users prefer having more interactive options to talk to the bot and have a less artificial conversation. These options include adding more variety to the pool of small talk options of the bot and including aside functionalities like jokes or simple games.
6. Users prefer getting information through examples with as much detail as possible.
7. Some users believe that the bot requires too much concretization in some instances, and find that behaviour a bit problematic. For example, users asked "how does packet fragmentation work?" and got an invalid concept request. This is the expected behaviour (the kind of IP fragmentation was not defined), but users think IPv4 class should be the default answer.
8. Users proposed a series of improvements that could be interesting to implement or investigate in the future:

- Include an "extend" option after every answer in order to provide a more profound explanation with more details if required.

- Provide external references to images, webpages, etc… to add complementary information.
- Add a command to display the bot's content map.
- Add a "/help" command to explain how the bot must be used.
- Have the option to quickly select a concept after making a spelling mistake detected by TeCePe. For example, if it suggested "Forwarding Table", only answer yes in order to perform the search again, rather than introducing everything again.

All this information will be taken into account when evaluating TeCePe's success and developing new functionalities in the future.

6. PROJECT'S PLANNING

6.1. Introduction

In this chapter we are going to define all aspects regarding bot's development phases, including all performed tasks with their correspondent starting and ending dates, as well as all economic aspects of the project.

6.2. Planning

All planned tasks regarding bot's conceptualization and development are shown in *Fig 5.1*, including duration time, starting and ending date for every task. The tasks are divided into smaller subtasks to have a more detailed view of how they were developed.

Task	Start Date	Duration (days)	End Date
State of the art analysis	10/09/2017	60	11/11/2017
Investigation about bot technologies	10/09/2017	14	24/09/2017
NPL and machine learning analysis	13/09/2017	5	17/09/2017
Programming approach and tools analysis	23/09/2017	17	10/10/2017
Bot platfrom analysis	07/10/2017	35	11/11/2017
Equipment setup	**10/11/2017**	**10**	**20/11/2017**
Telegram download and setup	10/11/2017	1	11/11/2017
Twitter download and setup	10/11/2017	1	11/11/2017
Python environment setup	11/11/2017	4	15/11/2017
Python libraries configuration	15/11/2017	5	20/11/2017
MySQL download and setup	18/11/2017	2	20/11/2017
First version development	**15/11/2017**	**74**	**23/01/2018**
Twitter connection test	15/11/2017	10	25/11/2017
Telegram connection test	23/11/2017	10	03/12/2017
Telegram-bot-api integration	01/12/2017	21	22/12/2017
Filtering function added	10/12/2017	5	15/12/2017
New handlers adition	22/12/2017	3	27/12/2017
Testing and bug correction	26/12/2017	28	23/01/2018
Second version development	**22/01/2018**	**34**	**25/02/2018**
Anytree tests	22/01/2018	3	25/01/2018
Tree structure definition	24/01/2018	10	03/02/2018
Anytree integration with the main code	02/02/2018	10	12/02/2018
Testing and bug correction	13/02/2018	12	25/02/2018
Third version development	**25/02/2018**	**16**	**13/03/2018**
Definition of MySQL tables	25/02/2018	3	28/02/2018
Implementation of MySQL in the main code	27/02/2018	5	04/03/2018
Testing and bug correction	04/03/2018	9	13/03/2018
Fourth version development	**14/03/2018**	**35**	**19/04/2018**
Tree loading method redesign	14/03/2018	5	19/03/2018
Delete obsolete functions	16/03/2018	5	21/03/2018
Query failure protocol addtion	18/03/2018	17	04/04/2018
Testing and bug correction	03/04/2018	16	19/04/2018
Final version development	**20/04/2018**	**24**	**14/05/2018**
Implementation of error correction	20/04/2018	12	02/05/2018
Improvement of query failure protocol	22/04/2018	6	28/05/2018
Testing and bug correction	27/04/2018	17	14/05/2018
Application deployment	**04/05/2018**	**15**	**19/05/2018**
Server configuration	04/05/2018	3	07/05/2018
Code adaptation	06/05/2018	8	14/05/2018
MySQL tables migration	13/05/2018	4	17/05/2018
Testing	17/05/2018	2	19/05/2018
Tests with users	**18/05/2018**	**17**	**04/06/2018**
Writting the report	**12/05/2018**	**30**	**12/06/2018**

Fig. 6.1 Planning table of the project

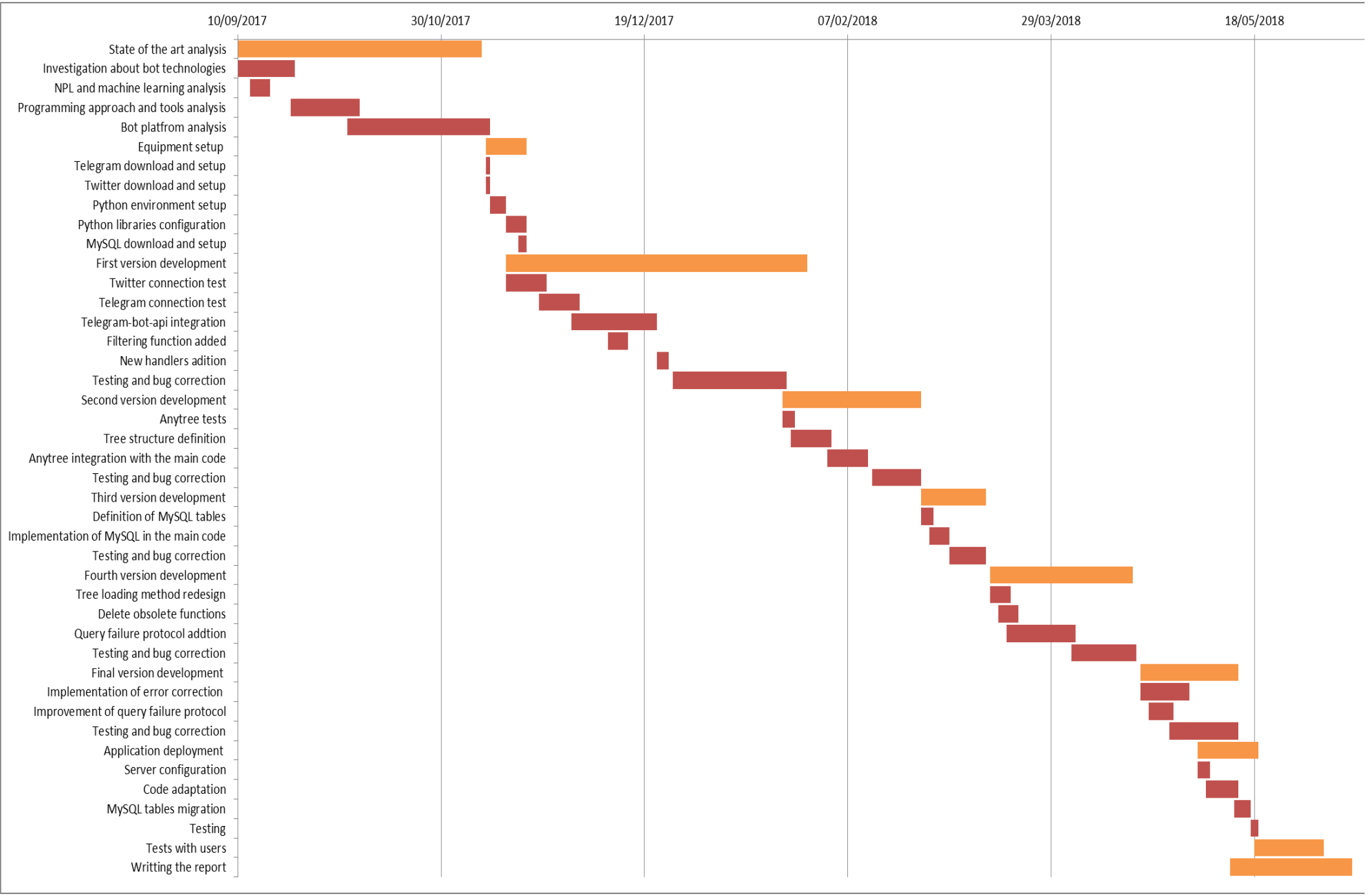

Fig. 6.2 Gantt diagram of the project

The previous image was a translation of the planning table into a Gantt diagram to provide a better visual representation of TeCePe's project. As it can be seen in both the table and diagram, the chosen strategy for the project's development has been dividing it into different versions that incorporated new functionalities on every iteration as well as fixing particular version issues in order to adapt the code to new changes. This is the reason why this planning is almost completely sequential, as it is necessary to make the previous version to implement new additions. However, both state of the art analysis and equipment setup allows some parallelization because they do not entirely depend on each other. The same can be said for user's testing and report writing.

6.3. Technical resources

In this section we will describe all equipment that had to be used during the life of the project, including both hardware and software required for development, testing and text formatting:

Hardware:

1. Smartphone Samsung Galaxy S8: Required for testing the bot.
2. Lenovo Personal Computer with Intel Core i5 and both Ubuntu and Windows 10 operative systems: Required for development, information searching and memory writing.
3. Remote UC3M Linux Server: Required to host the bot.

Software:

1. Python 2.7 environment: Mandatory for creating, compiling and executing the bot in Python.
2. Python libraries: Required for programming all TeCePe's algorithms and functionalities. Those libraries include:
3. Anytree
4. PyEnchant
5. Telegram-bot-api
6. NLTK
7. TextBlob
8. MySQLDB
9. MySQL and MySQL Workbench: Required for contacting and managing the database.
10. Microsoft Word: Used to write the project's report.
11. Microsoft Excel: Used to make all report tables and diagrams.

6.4. Economic analysis of the project:

All economic aspects of TeCePe's development will be described inside this section. Moreover, budget, direct costs indirect costs, and final cost summary will be included as well.

Economic environment

TeCePe has not been done with the intention of obtaining any economical revenue in mind. This is a tool that helps students during their learning process, so in theory it should not be used to get any economic value from using this bot. Therefore, TeCePe did not implement any way of obtaining economic benefits after its deployment.

However, it does not mean that, in the future, it could provide some economic benefits through its use. Although Telegram does not support "traditional add" revenue (using adds to be displayed on the application), adds could be added in other ways to generate profits. For example, when a user searches a concept, output the result plus some products related to that concept in order to incentivize users to buy these products. This way we would be including advertisements inside our bot along with getting some revenue from advertisers.

Direct costs

First of all, it is necessary to calculate the costs related to personnel. In this case, there have been two people working on the project: one engineer and one senior engineer, whose costs have been 20.50 € per hour and 33.00 € per hour respectively. The total amount has been calculated based directly on the number of hours worked in the project. The following table shows all details:

Table 6.1 PERSONNEL COST'S TABLE

Name and Surname	Category	Cost per hour (Euros)	Hours per month	Number of months	Total hours	Cost per month (Euros)	Total cost (Euros)
Soto, Ignacio	Senior Engineer	33.00	7.00	8.00	56.00	231.00	1.848.00
González Blázquez, Luis Félix	Engineer	22.50	37.50	8.00	300,00	1875,00	15.000,00
						Total	16.848,00

The amortization costs of all equipment used for the project have been calculated using the following equation:

$$\frac{A}{B} \text{ x C x D}$$

Fig. 6.3 Amortization's equation

A = Number of month since the date the equipment is used
B = Depreciation period
C = Equipment Cost (without VAT)
D = Percentage (%) of use dedicated to the project (usually 100%)

In the following table, all elements are presented with their correspondent costs. Notice that the server has no cost associated, as it was lent by UC3M for free:

Table 6.2 AMORTIZATIONS' COSTS TABLE

Description	Cost (Euros)	Percentage dedicated to the project	Months dedicated	Depreciation period	Incurred cost
Lenovo PC	480.00	100.00	8.00	60.00	64.00
Smartphone	250.00	100.00	6.00	60.00	25.00
External Server	0.00	1.00	2.00	60.00	0.00
				Total	89.00

The remaining costs are those that cannot be labelled as personnel or equipment costs:

Table 6.3 REMAINING DIRECT COSTS TABLE

Description	Cost per month (Euros)	Months	Incurred cost
Additional trips	20.00	8.00	160.00
		Total	160.00

Cost summary

Indirect costs have been calculated by multiplying the indirect cost tax with the sum of all direct costs. In our case, this tax has a 20% value:

Table 6.4 COSTS SUMMARY

Total cost budget (Euros)	Total cost budget (Euros)
Personnel	16.848
Amortization	89
Functioning costs	160
Indirect costs	3.419
Total	20.516

7. CONCLUSIONS

7.1. Objective fulfilment

The project's objective was developing an application that would help students and other people interested on learning about the TCP/IP protocol. In order to satisfy this scope, an original algorithm was created to search concepts using Language Processing to give the best possible answers to users. Its implementation was done in Python, dividing all programming in modules to separate efficiently all TeCePe's functionalities. Furthermore, TeCePe had a testing phase where all its functionalities and characteristics were evaluated by users to see the public's general satisfaction with the project.

In the beginning of this report, in section 1.2, we defined some objectives that TeCePe should meet in order to be considered a success. Therefore, we are going to analyse all of them taking into account TeCePe's development and users' feedback in order to later extract conclusions about TeCePe's overall success:

- The bot will must definitions of basic concepts about the TCP/IP protocols: This objective is the main idea behind TeCePe's conception. Developing a personalized algorithm not only helped fulfilling this requirement, but it has also been enhanced by using more advanced text processing techniques like syntax unit analysis. TeCePe is capable of providing many types of definitions based on what users asked for, and the proposed structure is simple to modify for adding new possible definition types. Therefore, we can conclude that TeCePe fulfilled this objective completely.
- The bot must be able to compare two concepts of the TCP/IP protocol, as long as they are compatible: TeCePe is able to detect if users are performing a comparison through text analysis. If so, then TeCePe will make that comparison, only if it is possible. Therefore, TeCePe also fulfilled this objective.
- The bot must be able to interpret users' intention to provide the best possible answer: As previously mentioned, TeCePe is able to analyse a sentence and obtain syntactical units in order to obtain the users intentions. Therefore, this objective is also fulfilled.
- The interface between the user and the bot must be clear and intuitive: TeCePe uses Telegram's application interface, which is very familiar among users (as Telegram is a popular application worldwide) and simple to use for every type of user. Therefore, TeCePe's interface is both clear and intuitive, fulfilling this objective.
- The answer's data must be stored in the most flexible way possible in order to simplify updates and modifications of the data: TeCePe uses a MySQL database in order to store all of its content due to both its popularity and simplicity. Moreover, all the information structures of the bot are not very complex, so making modifications or introducing new data is fairly easy and simple. However, to make any of those changes, managers require at least some basic database knowledge. Otherwise, managers cannot make any changes to the data,

which can compromise TeCePe's accessibility to some users. Therefore, this objective has been fulfilled, but there could be some room for improvement.

- The bot must interact as much as possible with users to simulate a human conversation and encourage interaction with the bot: TeCePe is able to receive any sentence from users and interpret intentions or concepts. Moreover, it is capable of performing some small talk with users (having answers for sentences like hello, thanks…) and detecting bad words (not answering questions in this last case, as it is considered impolite. Thus, it could be argued that TeCePe encourages interaction. However, as TeCePe tries to be very straightforward with answers, it seems to be artificial: users can clearly see that TeCePe is a bot based on how it replies to inputs. Therefore, this objective has been partially fulfilled, so some modifications should be done to improve TeCePe.
- The bot must deal with users spelling mistakes: TeCePe is able to detect spelling mistakes made by users on their inputs. These spelling mistakes will be detected only for those words related to TeCePe's internal data structure (as the rest are not relevant for its mechanisms). However, TeCePe does not correct those mistakes completely: the bot only tries to correct them and outputs all possible alternatives instead of performing the correction itself. Therefore, TeCePe does correct user's spelling mistakes, but it could be argued that more work could be done in order to make a full correction (correcting user's spelling mistakes instead of showing alternatives).

7.2. Project's general conclusions:

Based on all bot's development, user's feedback and fulfilment degree of the functionalities presented in the previous section of this chapter, we can obtain some conclusions about the degree of success of TeCePe:

- The proposed search algorithm for sentence analysis is very powerful, as it is able to retrieve sentence's ideas while respecting priorities among words. Having influence from the more traditional keyword searching algorithm, it is also fast and intuitive to use.
- TeCePe's internal data structure is well designed, as almost its entire core components are stored in an external database using intuitive (and simple) structures that can be modified with relative ease.
- TeCePe search engine is able to provide coherent and concise answers about TCP/IP protocols very quickly, making the bot very attractive for users that want to perform small searches about concepts regarding those protocols.
- TeCePe is hosted inside Telegram, which provides the bot with an attractive and familiar interface that users recognize and enjoy. It also helped releasing some workload of the programming of connection tasks of the bot.
- TeCePe manipulates human data and, in consequence, tries to perform a human conversation with users. However, it is too straightforward and users seem to miss some interactions to make it more appealing.
- TeCePe is able to detect mistakes inside users' sentences with high accuracy.

7.3. Future work

Having into account this is a software tool, our bot can always be updated to offer the best performance possible at all times, adapting its mechanisms to incorporate more advanced technologies or introducing new functionalities. Furthermore, some of these improvements spawned from development and/or user testing phases. Therefore, we are going to introduce some future improvements that could be investigated (and developed) to be incorporated later inside TeCePe.

Conversational module

We already saw that TeCePe can be considered as an "artificial bot", which means that it does not perform further interactions with users except delivering answers and some minor small talk. In consequence, creating a new module to manage other interactions with the bot besides the ones already implemented could be a good idea.

This module would get the remaining parts of input text, after performing the concept's search, and making a conversation like more traditional conversational bots do. It could get some inspiration from bots like ELIZA or ALICE for its core functionality.

This module is one of the most requested among users, as they all coincide that bots like TeCePe should try to be as human as possible to improve dialogue to be more fluent with users (a characteristic they usually find very attractive).

Text Correction module rework

Although TeCePe's text correction module works and its accuracy is sufficiently high, it is not perfect. TeCePe does not correct completely: it only outputs possible alternatives based on user's mistakes. Therefore, two improvements could be done in this case.

The first improvement would be performing an initial correction in first place in order to reduce the amount of alternatives or avoiding any conflict at all. This should have a much bigger module development however, as accuracy would need to be almost perfect and, although now it is pretty high, the tools available at this moment might not be enough to make it.

The second improvement was suggested by a couple of users. As TeCePe outputs alternatives, it could be interesting to let users select the alternative without having to write the sentence again with the correct word. Furthermore, if only one alternative is available, let users to text YES/NO to perform the correct search instead. It might not be too straightforward to program considering TeCePe's current architecture, but this functionality would improve its accessibility and, in some way, would make the bot more dynamic.

Extended information option

During users' testing phase, it was discovered that users find simple definitions not enough for bots of these characteristics. They favour synthesis and simplicity when search is more straightforward. However, in a lot of cases they usually want more details and explanations through images, examples… Therefore, one functionality that could be added is introducing a new command (or option) to, after receiving the answer, display more information related to the topic or answer. It could be provided either by the bot or redirecting to other webpages.

This improvement has similar implementation problems to the ones mentioned in the previous point, as TeCePe's current architecture might have problems with solutions that involve states (its structure does not have any). However, developing this would help users get more tools to know better the TCP/IP protocol, which is bot's ultimate objective.

Remote data management application

Although TeCePe's database tables are simple enough, it is mandatory to have at least some basic database knowledge. Otherwise, introducing new concepts or modifying them is too challenging to be done without it. This might be a problem for teachers that do not have experience with databases. Therefore, one solution to this problem could be developing an external application that would allow modifying that information through a more user-friendly interface. Hence, data managers would not require previous experiences with databases.

Its development does not seem to be very difficult at first glance, so it is an approachable improvement that would increase bot's quality. Moreover, the main node of the search tree should be able to be modified (which is now fixed) in order to allow TeCePe host more subject types.

8. BIBLIOGRAPHY

AISB. (2018). *Loebner Prize*. Retrieved June 9, 2018, from AISB website: http://www.aisb.org.uk/events/loebner-prize

Bunardzic, A. (2017, May 17). *Four Types Of Bots.* Retrieved May 9, 2018, from https://chatbotsmagazine.com/four-types-of-bots-432501e79a2f

Carpenter, R. (n.d.). *Cleverbot*. Retrieved June 1, 2018, from home of Cleverbot: http://www.cleverbot.com/

Chowdhury, G. G. (2018). Natural Language Processing. *European Journal of Education,* 160-175.

Deryugina, O. (2010). The history of chatterbots. *Scientific and Technical Information Processing, 37*(2), 143–147.

Giridhar, C. (2017). *Automate it!* Packt Publishing.

Holtgraves, T., & Han, T.-L. (2007). A procedure for studying online conversational processing using a chat bot. *Behavior Research Methods, 39(1),* 156-163.

IETF. (1973, January 21). *PARRY Encounters the DOCTOR.* Retrieved June 9, 2018, from IETF: https://tools.ietf.org/html/rfc439

Kelly, R. (2018, February 24). *pyenchant: Python bindings for the Enchant spellchecker*. Retrieved May 20, 2018, from gitHub: https://github.com/rfk/pyenchant

Kurose, J. F., & Ross, K. W. (2012). *Computer Networking: A Top-Down Approach.* Kansas: Pearson Education Limited.

Liddy, E. D. (2001). *Natural Language Processing.* New York: Marcel Decker, Inc.

Loria, S. (2018). *TextBlob: Simplified Text Processing*. Retrieved May 20, 2018, from TextBlob: http://textblob.readthedocs.io/en/dev/

Maruti Techlabs. (2017, November 7). *Top 5 Benefits Of Using Chatbots For Your Business.* Retrieved February 5, 2018, from Chatbot Magazine by OCTANE AI: https://chatbotsmagazine.com/top-5-benefits-with-using-chatbots-for-your-business-159a0cee7d8a

McNeal, M. L., & NewYear, D. (2013, November). Introducing Chatbots in Libraries. *Library Technology Reports*.

McTear, M., Callejas, Z., & Griol, D. (2016). Conversational Interfaces: A Brief History. In M. McTear, Z. Callejas, & D. Griol, *The conversational interface: talking to smart devices* (pp. 52-58). Switzerland: Springer International.

Michael, K. (2016). Science Fiction Is Full of Bots That Hurt People:... But these bots are here now. *IEEE Consumer Electronic Magazine, V*(4), 112-117.

Microsoft. (n.d.). *Language Understanding*. Retrieved June 9, 2018, from Microsoft Azure: https://azure.microsoft.com/es-es/services/cognitive-services/language-understanding-intelligent-service/

MuleSoft Inc. (2015, June 19). *What is an API? (Application Programming Interface).* Retrieved May 10, 2018, from Mulesoft: https://www.mulesoft.com/resources/api/what-is-an-api

NLTK Project. (2018, May 06). *NLTK 3.3 documentation*. Retrieved May 12, 2018, from NTLK Project: https://www.nltk.org/

Onur, V., Ferrara, E., Davis, C. A., Menczer, F., & Flammini, A. (2017, March 27). *William & Mary.* Retrieved February 2, 2018, from Online Human-Bot Interactions: Detection, Estimation, and Characterization: http://www.cs.wm.edu/~hnw/paper/tdsc12b.pdf

Python Software Foundation. (2001). *About us: Python*. Retrieved June 9, 2018, from Python: https://www.python.org/about/

SimilarWeb LTD. (2018, May). *Cleverbot.com Traffic Overview*. Retrieved June 1, 2018, from SimilarWeb LTD: https://www.similarweb.com/website/cleverbot.com#overview

Symantec. (n.d.). *What are bots?* Retrieved February 2, 2018, from Norton's website by Symantec: https://us.norton.com/internetsecurity-malware-what-are-bots.html

Technopedia Inc. (n.d.). *Artificial Intelligence (AI).* Retrieved June 3, 2018, from Technopedia: https://www.techopedia.com/definition/190/artificial-intelligence-ai

Telegram. (2018, February 13). *Telegram-Bot-API.* Retrieved June 08, 2018, from Telegram: https://core.telegram.org/bots/api

Telegram. (n.d.). *@BotFather.* Retrieved May 8, 2018, from Telegram: https://telegram.me/botfather

Toledo, L. (2015, July 8). *python-telegram-bot documentation.* Retrieved June 8, 2018, from python-telegram-bot: https://python-telegram-bot.readthedocs.io/en/stable/

Vegesna, A., Jain, P., & Porwal, D. (2018). Ontology based Chatbot (For E-commerce Website). *International Journal of Computer Applications, 179*(14), 51-55.

Wolchover, N. (2011, September 7). *How the Cleverbot Computer Chats Like a Human.* Retrieved February 4, 2018, from LiveScience: https://www.livescience.com/15940-cleverbot-computer-chats-human.html

Zeifman, I. (2017, January 24). *Bot Traffic Report 2016*. Retrieved February 2, 2018, from Imperva Incapsula: https://www.incapsula.com/blog/bot-traffic-report-2016.html

APPENDIX A: USERS' TEST TEMPLATE

In this appendix we will see the tests that users had to fill during TeCePe's testing phase. It is presented with Google's personal format:

TeCePe bot: testing and improvements on IPv4

In this form, you will be asked to "play" with TeCePe in order to evaluate how useful, interesting or special is, as well as suggesting ways that she can improve and become a better bot.

Initially, we will only ask her about concepts regarding the IPv4 routing protocol (in the future its knoledge will be increased, but for now this survey will only cover this concept).

Please, have in mind two things: this is still a project evolving over time, so changes could be done at any time without previous warnings. Secondly, be realistic with your improvements, as this can be sometimes limited due to time/budget constrains. Nevertheless, have fun!!

WARNING: She does not like people with bad language... so be polite!!

*Obligatorio

My experience with bots

1. **Please evaluate the following statements from 1 to 5, being 1 complete disagreement and 5 total agreement.** *

Marca solo un óvalo por fila.

	1	2	3	4	5
When requesting for definitions/concepts, I prefer having the most ammount of details possible	◯	◯	◯	◯	◯
When talking to bots, I prefer to be answered as humanly as possible	◯	◯	◯	◯	◯
I want bots to have entertaining functions beside its main purpose (jokes, conversations...)	◯	◯	◯	◯	◯
I prefer examples over definitions	◯	◯	◯	◯	◯
Bots like TeCePe could help me studying a subject	◯	◯	◯	◯	◯
I had previous experiences with chatbots before	◯	◯	◯	◯	◯
I think chatbots can provide useful functionalities to help me in my work/studies	◯	◯	◯	◯	◯
I prefer bots to be integrated inside popular applications (Telegram, Twitter...) rather than being a separate enitity	◯	◯	◯	◯	◯

Fixed questions

We will ask TeCePe some questions. Please evaluate the following statements from 1 to 5, being 1 the lowest possible ranking and 5 the best one.

2. **Hi TeCePe: what is packet forwarding?** *

Marca solo un óvalo por fila.

	1	2	3	4	5
The question was answered correctly					
The definition was clear and provided the information I required					
The search was fast					

3. **How does packet fragmentation work?** *

Marca solo un óvalo por fila.

	1	2	3	4	5
The question was answered correctly					
The definition was clear and provided the information I required					
The search was fast					

4. **What is the difference between packet routing and packet forwarding?** *

Marca solo un óvalo por fila.

	1	2	3	4	5
The question was answered correctly					
The definition was clear and provided the information I required					
The search was fast					

5. **Where is the link state algorith used?** *

Marca solo un óvalo por fila.

	1	2	3	4	5
The question was answered correctly					
The definition was clear and provided the information I required					
The search was fast					

6. **Compare unicast routing with multicast routing** *

Marca solo un óvalo por fila.

	1	2	3	4	5
The question was answered correctly					
The definition was clear and provided the information I required					
The search was fast					

Time to talk!!

Please take some time to talk with TeCePe without restrictions (have in mind it is not avalaible for dating/marriage...). I reccomend taking 5-10 min, but it is up to you. Afterwards, fill the next text boxes with as much details as possible.

7. **Specify the probems/issues you encountered while talking with TeCePe, or possible concerns you might have.**

8. **Which improvements would you like to see implemented in TeCePe?** *

APPENDIX B: DATABASE CONTENT

In this appendix we will see all contents inside TeCePe at the moment of finishing this report. It must be taken into account that its knowledge might change and get bigger. All definitions and information inside this annex has been obtained from (Kurose & Ross, 2012).

Each concept has an entry, which is the answer outputted to the user, depending on the context. If no text is on any of the fields, it means an error message will be sent instead.

IP

- **What:** IP (Internet Protocol) is a communication protocol of the network layer. It has two possible versions: IPv4 and IPv6.
- **How:**
- **Where:** IP protocol was defined in RFC 791 in 1981. It is used in most network-oriented applications all over the world.
- **When:** IP is used when some data must be transmitted through a Network-Layer based network.
- **Compare with []:**

Forwarding

- **What:** Forwarding is packet's transfer from router's incoming interface to the appropriate outgoing interface.
- **How:** Forwarding is done by the router using its forwarding table.
- **Where:** Forwarding is used in network protocols like IPv4 and IPv6.
- **When:** Forwarding is used when a packet must be transferred from router's ingress interface to its corresponding egress interface.
- **Compare with [routing]:** Forwarding is transferring one packet from an input interface to its corresponding output interface. Routing is designating the path taken by those packets from a sender to a receiver.

Forwarding Table

- **What:** Forwarding table is a table used by the forwarding function of a router to transfer a packet from its input interface to the corresponding output interface.
- **How:** A router examines arriving packet's header to map its interface inside the router's forwarding table. This value will be selected as output when new

incoming packets want to go to its source router/host, as this value indicates the next link that packets use to reach its destination.

- **Where:** It is stored inside the forwarding control plane of a router.
- **When:** It is used in packet forwarding.
- **Compare with [routing table]:** The forwarding table is used in packet forwarding, while routing table is used in packet routing.

Routing

- **What:** Routing is designating the path that a packet will traverse from a sender to its corresponding receiver.
- **How:** A router uses the routing table to check which is the next-hop of an incoming packet.
- **Where:** Routing is used in routing algorithms of Network-layer protocols.
- **When:**
- **Compare with [forwarding]:** Forwarding is transferring one packet from an input interface to its corresponding output interface. Routing is designating the path taken by those packets from a sender to a receiver.

Routing Table

- **What:** The routing table is a table used by the router's control plane to determine the path that a packet will traverse to reach its destination.
- **How:** Using, or more, routing protocols, the router creates the table with two columns: packet’s destination network and next-hop address. When a packet enters through an input interface, the router checks its destination and decides its next-hop based on the registered entries of the table.
- **Where:** The routing table is located inside router’s control plane. Each entry is composed of one destination network and one next-hop address.
- **When:** It is used in packet routing.
- **Compare with [forwarding table]:** The forwarding table is used in packet forwarding, while routing table is used in packet routing.

Hierarchical

- **What:** Hierarchical routing consists of arranging routers in a hierarchical structure to help with scalability and administrative autonomy's problems.
- **How:** Hierarchical routing is performed by the organization of routers in Autonomous Systems (AS), who have the same administrative control and run the same routing algorithm (intra autonomous system routing protocol). In order to communicate with other AS, they need routers called gateway routers to perform connections using inter autonomous system protocols.
- **Where:** It is used inside networks where topology's complexity must be reduced and/or the network requires less congestion.
- **When:** The use of hierarchical routing varies depending on the desired scalability and administrative control.
- **Compare with []:**

Distance Vector

- **What:** Distance Vector algorithm is a distributed (communication between neighbour routers), iterative (continues until no more information is exchanged) and asyncronous (routers do not need to operate in lockstep with each other) algorithm used to calculate the minimum distance between routers of a network.
- **How:** By applying the Bellman-Ford equation dx(y) = minv[c(x,y) + dv(y)],b the router calculates the minimum distance to the nodes of the network (dx) based on the information provided by all its neighbour nodes.
- **Where:** Distance Vector algorithms are used in protocols based on decentralized routing algorithms, where message exchanges occur between neighbour routers.
- **When:**
- **Compare with [Link state]:** For distance vector algorithms, it is not necessary to have whole network information, as the routers only exchange messages between neighbours. On the other hand, link state requires the knowledge of the whole network (routers talk between all elements of the network).

Link State

- **What:** Link state is a global routing algorithm that computes the least-cost path between a source and a destination by having total knowledge of the network.
- **How:**
- **Where:** Link State algorithms are used in protocols based on centralized routing algorithms, where message exchanges occur between all routers of the network.

- **When:** Link State algorithms are used in protocols based on message exchanges between all routers of a network, in order to calculate the minimum distance between a source and a destination.
- **Compare with [Distance Vector]:** For distance vector algorithms, it is not necessary to have whole network information, as the routers only exchange messages between neighbours. On the other hand, link state requires the knowledge of the whole network (routers talk between all elements of the network).

Unicast

- **What:** Unicast routing occurs when a packet must be sent to only one destination.
- **How:** In order to use unicast routing, the packet must be sent to one unique destination with one unicast IP address.
- **Where:**
- **When:** It is when the receiver of the information is a single device/router.
- **Compare with [Multicast]:** Unicast routing is used when the destination of a packet is unique, while multicast routing is used when the packet must go to more than one destination set of destinations.

Multicast

- **What:** Multicast routing occurs when a packet must be sent to a set of network nodes using a single address.
- **How:** In order to use multicast routing, a packet must have a multicast address for its destination address. While traversing through the network, routers will redirect these packets to multiple destinations.
- **Where:**
- **When:** It is used when receivers of the information are a set of routers/hosts.
- **Compare with [Unicast]:** Unicast routing is used when the destination of a packet is unique, while multicast routing is used when the packet must go to more than one destination set of destinations.

IPv4

- **What:** IPv4 is a connectionless protocol used on packet-switched networks. It operates on a best-effort model (packet delivering is not guaranteed).
- **How:**
- **Where:** IPv4 can be found in almost every device/technology connected to the Internet.
- **When:** IPv4 is used when an application/technology requires packet transmission through a packet-switched based network.
- **Compare with [IPv6]:** IPv6 is the evolution of the IPv4 protocol. Therefore, it incorporated new functions like expanding addressing capabilities, a streamlined 40-byte header, and flow labelling and priority. IPv6 also introduces the Anycast direction, which allows a datagram to be delivered to any one of a group of hosts.

Datagram

- **What:** Datagrams are the basic unit of network layer protocols. They are composed of data from previous layers plus a header that provides information about the IPv4 details.
- **How:** Datagrams in IPv4 are composed of 32 bit words divided in different fields regarding protocol information (addresses, flags…).
- **Where:** Routers use datagrams in network-layer applications.
- **When:**
- **Compare with [ipv6 datagram]:** IPv4 datagrams are used in IPv4 routing, while IPv6 datagrams are used in IPv6 routing.

Datagram format

- **What:** The format of an Ipv4 datagram is the following: Version Number (4 bits), Header Length (4 bits), Type of Service (8 bits), Datagram Length (16 bits), Identifier (16 bits), Flags (3 bits), Fragmentation Offset (13 bits), Time To Live (8 bits), Upper-Layer Protocol (8 bits), Header Checksum (16 bits), Source IP address (32 bits), Destination IP address (32 bits), Options (32 bit words, optional) and data from previous layers.
- **How:** The host that sends the packet fills the headers. Only those devices that can manage network level applications are allowed to read and/or modify datagram's header content.
- **Where:**
- **When:**

- **Compare with [ipv6 datagram format]:** Both IPv4 and IPv6 datagrams use 32 bit words. However, the main differences between both rely on the changes introduced for IPv6: expanded addressing capabilities, streamed-line 40 byte header and flow labelling and priority.

Fragmentation

- **What:** Routers produce fragmentation when a packet must be divided into smaller units to be transmitted correctly. It is done when the link's MTU is smaller than the length of the transmitted packet.
- **How:** At certain parts of the network, MTU requirements for links may change due to a large variety of reasons. Therefore, there could be some instances where packets will be too large to be transmitted through a link. Hence, routers can perform a datagram fragmentation to divide the packet into smaller chunks that will fulfil the transmission requirements. In order to keep all fragments in order, the Fragment Offset and Flag fields are modified for the destination host (responsible of reassembling the packet) to check in which byte of the original datagram the fragment belongs to.
- **Where:** Fragmentation is done inside routers that support this functionality.
- **When:** Fragmentation is required when a packet is larger than the MTU of a link.
- **Compare with []:**

Fragment

- **What:** A fragment is a datagram's piece created from fragmentation. The addition of all fragments composes the original datagram.
- **How:** When fragmentation is done, a packet is divided into two or more smaller fragments that will be encapsulated in different link-layer frames and sent to the destination host, where they will be combined to reproduce the original datagram.
- **Where:** Fragments are created inside routers that support packet fragmentation.
- **When:** Fragments are created when packet fragmentation is required.
- **Compare with[]:**

Reassembly

- **What:** Reassembly is the process where datagram fragments are combined to create the original datagram that had to be fragmented.
- **How:** When a final host receives a datagram, it checks on its header fields if it corresponds to a fragment (fragment offset, flag and ID). If so, it will use the fragmentation offset field to check which byte of the original datagram corresponds to, and combine all of them to reproduce the original datagram.
- **Where:** Reassembly is always done in the final host, and never in routers of datagram's path.
- **When:** Reassembly is necessary if received data corresponds to a set of fragments to produce the original message.
- **Compare with []:**

IPv4 Addressing

- **What:** Addressing in a network is giving an address to a router or host interface in order to have a unique identification to reach other networks and to be reached for transmitting/receiving information. This address must be contained inside the subnet which the network belongs to.
- **How:** Addressing can be done manually through an interface direct configuration or with the help of technologies like DHCP.
- **Where:**
- **When:** IPv4 is necessary for all applications that require IPV4 connectivity.
- **Compare with[IPv6 Addressing]:**

IPv4 Address

- **What:** An IPv4 address is a 32-bit long unique identifier used in Datagram IPv4 Routing.
- **How:** They are usually represented in dotted-decimal notation, where each byte of the address is separated by a period. The total number of addresses that can exist is 2^{32}.
- **Where:** They are used in IPv4 addressing.
- **When:** They are used in IPv4 addressing.
- **Compare with [IPv6 Address]:** IPv4 addresses are 32 bits long and used in IPv4 routing, while IPv6 addresses are 128 bits long and used in IPv6 routing.

Subnetting

- **What:** Subnetting is dividing a set of IPv4 addresses in smaller subsets.
- **How:** Having a set of addresses and a netmask, we can create more subnets by changing each time the next bit of the subnet bit until we arrive to the expected division. For example, if we have the subnet 134.0.0.0/24, we can now divide this one into two different subnets: 134.0.0.0/25 (134.0.0.0000000) and 134.0.0.128/25 (134.0.0.10000000), obtaining two different subnets where we can address in different networks. In order to check how many hosts a subnet can have, we need to subtract the netmask from 32 and another two (all 0's in the host part represent the network and all 1's represent the broadcast direction). For example, in the network 134.0.0.0/25 we can have 2 to the power of (32-25) -2 hosts (addresses) available.
- **Where:**
- **When:** Subnetting is necessary in classless routing, because sometimes networks do not require many addresses and, using this method, the available address space is better distributed (specially having into account IPv4 has limited addressing space available).
- **Compare with []:**

IPv4 Assigment

- **What:** IPv4 assignment consists of setting a pool of addresses to an organization for addressing.
- **How:** In subnet addressing, the 32-bit IP address is divided into two parts: the host part and the network part. The netmask (or network prefix) determines the amount of bits that belong to the address network part (starting from the most significant bit). Usually, every organization is assigned a block of contiguous addresses for its addressing, which can also be subnetted if needed.
- **Where:**
- **When:**
- **Compare with []:**

CRDI

- **What:** Classless Interdomain Routing is an addressing assignment strategy that allows using non-fixed netmasks in block assignment of IP addresses.
- **How:** Netmask can get any value from 0 to 32 bits, which allows flexibility to have the best subnet for every case.

- **Where:**
- **When:** CRDI is used in almost every network assignment policies across the Internet.
- **Compare with [Classless]:** In classful assignment, netmasks have fixed values depending on the class, while in CRDI these values can go from 0 to 32 without distinction.

Classful

- **What:** Classful routing is an addressing assignment strategy that allows using fixed netmasks in block assignment of IP addresses.
- **How:** There are a set of classes in which portions of IP addresses can be. These have different values for each fixed netmasks: 8, 16 and 24 (class A, B and C respectively). Therefore, the amount of hosts that this subnets store are fixed, which proved to be inefficient due to being either too large (wasting valuable space) or too small.
- **Where:**
- **When:** Classful routing is being abandoned for not being too flexible with its assignment strategy compared to CRDI routing.
- **Compare with [CRDI]:** In classful assignment, netmasks have fixed values depending on the class, while in CRDI these values can go from 0 to 32 without distinction.

DHCP

- **What:** DHCP (Dynamic Host Configuration Protocol) is a protocol that allows a host to obtain an IP address automatically.
- **How:** A host obtains an IP using DHCP by firstly sending a DHCP discover message (UDP packet with port 67 and Broadcast IP 255.255.255.255 as destination) with source IP 0.0.0.0 (this host). After receiving it, the server will broadcast a DHCP offer message containing the transaction ID, proposed IP address and address lease time. Once the host receives this message, it will send a DHCP request echoing back the configuration parameters to the DHCP server, which will respond with a DHCP ACK, finishing the allocation. This IP can be configured to be both temporal and selected each time the host connects to the same network.
- **Where:**
- **When:**
- **Compare with []:**

NAT

- **What:** NAT (Network Address Translation) is a mechanism to remap one IP address space into another in real time. It is used for saving IP address space, but it can also provide further security measures to users inside a private network.
- **How:** A router that allows NAT-Translation must have at least two interfaces: one with a public IP to connect with the rest of the Internet and another one connected to a realm with private addresses. All packets whose destination is one of the "private devices" will always use the public IP (never the private one). The router will translate that public IP to its corresponding private destination and vice versa (translating form private addresses to the corresponding public IP address) by changing the destination (source) IP and destination (source) port in its NAT table.
- **Where:** NAT is used in SOHO subnets.
- **When:**
- **Compare with []:**

NAT Table

- **What:** NAT table keeps record of the translation from private to public addresses, and vice versa, in the NAT mechanism of a router.
- **How:** Each entry of the table is composed of four elements: a public IP address, a private IP address, and their corresponding ports. Each time a packet wants to exit the private network, the router selects a free port and replaces the source port with this new one, keeping the assignment for the incoming datagrams for packets destined to that private address.
- **Where:**
- **When:** The NAT Table is used by routers in NAT.
- **Compare with []:**

UPnP

- **What:** UPnP (Universal Plug and Play) is a protocol that allows a host to discover and configure a nearby NAT.
- **How:** If both the host and NAT are compatible, an application running in a host can request a NAT mapping between its private IP address and port number and the public IP address and port number. If the NAT accepts the request, it creates the mapping, allowing nodes from outside to generate TCP connections.
- **Where:**

- **When:** UPnP is used in networks where NAT is avalaible.
- **Compare with []:**

ICMP

- **What:** ICMP (Internet Control Message Protocol) is used by hosts and routers to communicate network-layer information to end systems.
- **How:** Depending on the application, an ICMP message with be sent with a certain type and a code. Other systems can reply to this message with other ones with different types and codes.
- **Where:** ICMP is used in many network-oriented applications, for example ping, traceroute...
- **When:**
- **Compare with []:**

ICMP message

- **What:** An ICMP message is a message used by the ICMP protocol. It is usually composed of a ICMP type and a code.
- **How:** ICMP messages are generated by a host/router and added to a datagram as data from an upper layer. These have an ICMP type and a code, and its combination have a certain meaning regarding the connection to be interpreted by an application or router. For example, an ICMP message with Type = 0 and code = 0 is an echo reply (to a ping).
- **Where:** ICMP messages are created when using certain network-oriented applications.
- **When:** ICMP messages are used in the ICMP Protocol..
- **Compare with []:**

IPv6

- **What:** IPv6 is the evolution of IPv4 network-layer protocol to solve the lack of addressing space.
- **How:**
- **Where:** IPv6 can be found in almost every device/technology connected to the Internet.
- **When:** IPv6 is used when an application/technology requires packet transmission through a packet-switched based network that supports IPv6.

- **Compare with [IPv4]:** IPv6 is the evolution of the IPv4 protocol. Therefore, it incorporated new functions like expanding addressing capabilities, a streamlined 40-byte header, and flow labelling and priority. IPv6 also introduces the Anycast direction, which allows a datagram to be delivered to any one of a group of hosts.

IPv6 Datagram

- **What:** Datagrams are the basic unit of network layer protocols. They are composed of data from previous layers plus a header that provides information about the IPv6 details.
- **How:** Datagrams in IPv6 are composed of 32 bit words divided in different fields regarding protocol information (addresses, flags…).
- **Where:** Routers use IPv6 datagrams in network-layer applications that support IPv6.
- **When:**
- **Compare with [ipv6 datagram]:** IPv4 datagrams are used in IPv4 routing, while IPv6 datagrams are used in IPv6 routing.

Datagram format

- **What:** The format of an Ipv6 datagram is similar to the IPv4 datagram (both use 32 bit words) but with different fields: the following: Version Number (4 bits), Traffic Class (8 bits), Flow Label (20 bits), Payload Length (16 bits), Next hdr (8 bits), Hop Limit (8 bits), Source Address (128 bits, four words), Destination Address (128 bits, four words) and data from previous layers.
- **How:** The host that sends the packet fills the headers. Only those devices that can manage network level applications are allowed to read and/or modify datagram's header content.
- **Where:**
- **When:**
- **Compare with [ipv6 datagram format]:** Both IPv4 and IPv6 datagrams use 32 bit words. However, the main differences between both rely on the changes introduced for IPv6: expanded addressing capabilities, streamed-line 40 byte header and flow labelling and priority.

IPv6 Addressing

- **What:** Addressing in a network is giving an address to a router or host interface in order to have a unique identification to reach other networks and to be reached for transmitting/receiving information. In IPv6, this addressing works differently from IPv4, as Ipv6 addressing is much more automatic than "traditional" IPv4 addressing.
- **How:** Regional registries are assigned a set of IPv6 prefixes to share with their clients.
- **Where:**
- **When:** IPv6 is necessary for all applications that require IPV6 connectivity.
- **Compare with[IPv6 Addressing]:**

TCP

- **What:** TCP is an Internet transport-layer, connection-oriented, reliable transport protocol. It provides functions like error correction, retransmissions and cumulative acknowledgments. It is said to be connection-oriented because before one application process can begin to send data to another, the both need to "handshake" with each other first. It provides a reliable data transfer service, ensuring all segments will be received by a hots uncorrupted, without gaps, without duplication and in sequence.
- **How:**
- **Where:** TCP is used by connection-oriented applications.
- **When:**
- **Compare with [IP]:** IP is a network-layer protocol with best-effort service model, while TCP is a transport-layer protocol with reliable data transfer service.

Segment

- **What:** A segment is the basic unit of the transport-layer. It is composed of a header, divided in fields, and data from previous layers.
- **How:** TCP segments are composed of 32 bit words divided in different fields regarding the connection information (addresses, flags…) plus the data from previous layers.
- **Where:** Segments are used in applications that use TCP connection.
- **When:**
- **Compare with []:**

Segment Format

- **What:** The format of a TCP segment is the following: Source Port (16 bits), Destination Port (16 bits), Sequence Number (32 bits), Acknowledgment number (32 bits), Header Length (4 bits), Unused (3 bits), Control flags (9 bits), Receive Window (16 bits), Internet Checksum (16 bits), Urgent Data Pointer (16 bits), Options (optional 32 bit words) and data from previous layers.
- **How:** The host that sends the segment fills the headers. Only those devices that can manage transport level applications are allowed to read and/or modify datagram's header content. The two most important headers are the Sequence number (byte-stream number of the first byte in a segment, and the Acknowledge number.
- **Where:**
- **When:**
- **Compare with []:**

Round-Trip Time

- **What:** Round-trip time is the amount of time that a packet uses in order to travel from one host to another and back. TCP uses this RTT to calculate segment retransmission timeouts.
- **How:** TCP has to estimate the RTT of packets in order to calculate the timeout interval of retransmissions. To do so, TCP calculates the average using the following formula: EstimatedRTT = (1-alpha) * EstimatedRTT + alpha * SampleRTT, being SampleRTT the RTT of the previous segment sent and alpha a fluctuation value from 0 to 1. For calculating the Standard Deviation, the formula is: DevRTT = (1 – beta) * DevRTT + beta * |SampleRTT – Estimated RTT|
- **Where:** Round Trip Time is used in TCP to calculate the Estimated RTT segment time.
- **When:**
- **Compare with []:**

Timeout Interval

- **What:** Timeout in TCP is the amount of time required to produce a retransmission. In TCP varies depending on segment's RTTs.
- **How:** To calculate the timeout interval, it is necessary to apply the formula TimeoutInterval = Estimated RTT + 4*DevRTT. When this timeout expires after sending a segment, if no ACK was received, the segment will be sent again and the timer will restart.
- **Where:** Timeout is used by TCP to calculate the amount of time is required before doing a retransmission.
- **When:** Timeout is used by TCP to calculate the amount of time is required before doing a retransmission.
- **Compare with []:**

Flow Control

- **What:** TCP provides a flow control service to eliminate the possibility that a sender overflows receiver's buffers.
- **How:** In order to provide flow control, each TCP sender uses a receive window to keep control of the amount of packets sent at each time.
- **Where:**
- **When:**
- **Compare with []:**

Connection management

- **What:** TCP provides a flow control service to eliminate the possibility that a sender overflows receiver's buffers.
- **How:** In order to provide flow control, each TCP sender uses a receive window to keep control of the amount of packets sent at each time.
- **Where:** It is used in TCP connections.
- **When:**
- **Compare with []:**

APPENDIX C: DEPLOYMENT OF TECEPE

In this appendix we will describe how TeCePe was deployed in the remote UC3M server using some figures as visual support.

First of all, it is necessary to upload all files required for TeCePe from the local PC to the desired folder of the remote server. In order to do so, we have to use the command *scp* (secure copy) through an shh connection. Therefore, after executing the command, it is mandatory to introduce user's password.

In both *Fig. A.1* and *Fig A.2* we can see the commands used to translate TeCePe's definitive version and its wordlist to the remote server:

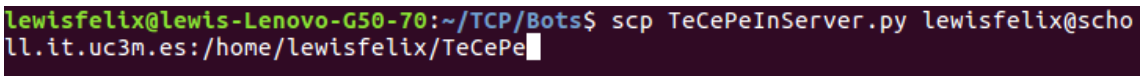

Fig. A.1

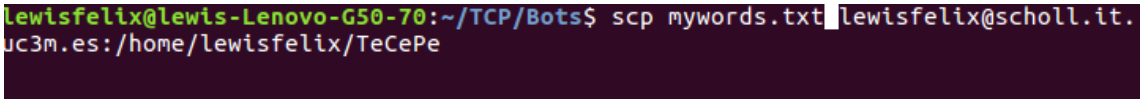

Fig. A.2

To check that our files were correctly uploaded, we will access the server with a ssh connection, as seen in *Fig A.3*:

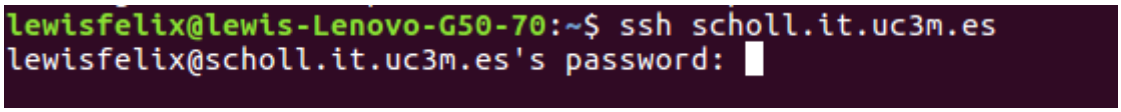

Fig. A.3

In the server, we need to go to the folder where TeCePe has been copied to and use the command *ls* (list directories) to check that both are inside, as shown in *Fig A.4*:

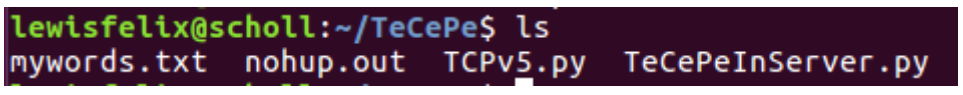

Fig. A.4

It is necessary to export the PC's local database to import it inside the remote server using *mysqldump* to generate a *.sql* archive, which includes some MySQL commands to copy tables inside a database. The fields of the command are the following: -u [username of the database] –p (introduce password afterwards) [Database name] [Table] (can be omitted if all) > [name of the new file]. Dsfghj shows how to use this command in the local PC:

```
lewisfelix@lewis-Lenovo-G50-70:~/TCP/Bots$ mysqldump -u root -p TCPIP Key_words
> key_words_good.sql
```

Fig. A.5

This archive must be copied to the remote server as in *Fig A.6*:

```
lewisfelix@lewis-Lenovo-G50-70:~/TCP/Bots$ scp key_words_good.sql lewisfelix@sch
oll.it.uc3m.es:/home/lewisfelix/TeCePe
```

Fig. A.6

In the remote server, to import the tables, it is required to perform the command shown in *Fig. A.7* with the fields: -u [username] –p [server's database] < [file to import name]:

```
lewisfelix@scholl:~$ mysql -u tecepebot -p tecepebotdb < key_words_good.sql
```

Fig. A.7

To run TeCePe as a background process inside the server (we want it to be running at all times, even after closing the ssh connection) we need to use the command shown in *Fig A.8*. The output of the program will be redirected to a *nohup.out* file if no other entry name was given.

```
lewisfelix@scholl:~/TeCePe$ nohup python TeCePeInServer.py &
```

Fig. A.8

APPENDIX D: LEGAL FRAMEWORK

In this appendix we are going analyse all legal restrictions that TeCePe is summited to as well as all aspects regarding intellectual property about TeCePe's content.

Users' Privacy

TeCePe has access to users' information regarding their personal information about their Telegram accounts. As shown in chapter 4, TeCePe uses Telegram Updates to retrieve the chat where the text of each update is stored. These updates hold information about user's account, which could be a thread to their privacy if certain personal details of the user could be seen using these updates, like its telephone number. Fortunately, Telegram updates do not hold information that is not public to every user. Therefore, TeCePe will never have the ability to retrieve private details from users (like their telephone number). Hence, TeCePe will always respect user's privacy when using Telegram's update system to not thread their privacy at any moment.

Information veracity

Although TeCePe has been developed for the TCP/IP protocols, and having as its main source of information the trustful Kurose's book (Kurose & Ross, 2012), its definitions and answers should never substitute any official information or teacher's advice at any moment. Therefore, users are advised when starting the bot that TeCePe should never be their main source of information and, in consequence, it is recommended to contrast it with other sources.

Bot's ownership

It is important to analyse if storing the bot in an external platform might compromise its ownership. In TeCePe's case, as it is hosted inside Telegram's application, Telegram might own the bot instead of the developer. Therefore, this fact should always be taken into account when developing the bot.

In this case, Telegram only provides the platform and the tools to create the connection with Telegram. However, bot's ownership always corresponds to the developer. Therefore, Telegram cannot ask at any point to have the bot code or claim to be bot's proprietary. Nevertheless, if one bot is suspected to be malicious or infringe Telegram's license agreement, it could stop providing its service to the bot and might ban the developer from their platform.