Compressor/encryptor module design for datacenters

Author:
Ivan Martínez Pérez

Grado en Ingeniería Informática
Especialidad en Ingeniería de Computadores

26 de Junio de 2014

Director:
Luis Domingo Velasco Esteban, Arquitectura de computadores

Facultad de Informática de Barcelona
Universidad Politécnica de Cataluña (UPC) - BarcelonaTech
Resumen

En el siguiente trabajo veremos los diferentes pasos seguidos para llegar a la implementación final de nuestro módulo.

Para ello y en primer lugar, el lector encontrará toda aquella información necesaria para adquirir una base general sobre los principales temas que trataremos, como la compresión o la encriptación.

Seguidamente dos diseños serán propuestos, uno hardware, el cual es puramente teórico y nos sirve de base para el diseño software, el cual si ha sido implementado y se analizará más en detalle. También se llevarán a cabo algunos análisis de parámetros referentes al rendimiento de nuestro módulo.

Finalmente se expondrá la planificación general de nuestro proyecto junto a las conclusiones de este.

Resum

Al següent treball veurem els diferents passos seguits per arribar a la implementació final del nostre mòdul.

Per això y en primer lloc, el lector trobarà tota aquella informació necessària per adquirir una base general sobre els principals temes que tractarem, com la compressió o l’encriptació.

Seguidament dos dissenys seran proposats, un hardware, el qual és purament teòric i ens serveix de base per al disseny software, el qual si ha estat implementat i s’analitzarà més detalladament. També es realitzaran alguns anàlisis de paràmetres referents al rendiment del nostre mòdul.

Finalment s’exposarà la planificació general del nostre projecte juntament amb les conclusions d’aquest.

Abstract

In the next work we will see all followed steps to implement our final module.

At first place, reader found all those necessary information to gain the general base about main themes that we will treat, like compression or encryption.

Straightaway two designs will be proposed, a hardware design, which is purely theoretical and will be use like base for software design, which has been implemented and will be analyzed in detail. Also will be made some performance analysis of different module parameters.

Finally, the general project planning will be exposed, together with our conclusions of this project.
Contents

Resumen 1

Prefacio 7

1 Introduction 8
1.1 Origins 8
1.2 Motivation and Objectives 8
1.3 Actors 9
1.3.1 Project Director 9
1.3.2 Developer 9
1.3.3 Datacenters 9
1.4 Possible obstacles and solutions 9
1.4.1 Inability to access real datacenter 9
1.4.2 Bugs 9
1.4.3 Hardware vs Software 9
1.5 Working Tools 10
1.5.1 Git 10
1.5.2 Netbeans IDE 10
1.5.3 Routers 10
1.5.4 PC 10
1.6 Work Method 10
1.7 Validation Method 11

2 State of the art 12
2.1 Network 12
2.1.1 IP Compression 12
2.1.2 IP Encryption 13
2.2 Compression 14
2.3 Encryption 15
2.4 Unifying Concepts 15

3 Hardware Design 17
3.1 Proposed Implementation 17
3.1.1 Compression Module 17
3.1.2 Encryption Module 19
3.1.3 Network Module 21
3.2 Unified Design 23
4 Software Design

4.1 AES
   4.1.1 AddRoundKeys
   4.1.2 SubBytes
   4.1.3 ShiftRows
   4.1.4 MixColumns

4.2 LZSS

4.3 Network Manager

4.4 Core Manager
   4.4.1 Session
   4.4.2 Buffers Controller

5 Software Evaluation

5.1 Compression Unit Test
   5.1.1 Times
   5.1.2 Compression Ratio

5.2 Load Test

6 Project Management

6.1 Temporal Planning
   6.1.1 Planning
   6.1.2 Preparation
   6.1.3 Implementation and testing
   6.1.4 Presentation
   6.1.5 Resources
   6.1.6 Assessing alternatives and action plan

6.2 Economic Planning
   6.2.1 Planning
   6.2.2 Preparation
   6.2.3 Implementation and Testing
   6.2.4 Presentation
   6.2.5 Total Cost

6.3 Sustainability

7 Conclusions

Acknowledgements
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Compressor basic design</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>Compressor parallel design</td>
<td>18</td>
</tr>
<tr>
<td>3.3</td>
<td>Encryptor Basic design</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>Encryptor Parallel design</td>
<td>20</td>
</tr>
<tr>
<td>3.5</td>
<td>Key generation on hybrid cryptosystems through Diffie-Hellman key exchange</td>
<td>21</td>
</tr>
<tr>
<td>3.6</td>
<td>Pipelining over AES transformation functions</td>
<td>22</td>
</tr>
<tr>
<td>3.7</td>
<td>Overall hardware design block diagram</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>AES algorithm structure</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Rijndael’s key schedule pseudocode</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>S-box: substitution values for the byte xy (in hexadecimal format)</td>
<td>26</td>
</tr>
<tr>
<td>4.4</td>
<td>ShiftRows operation</td>
<td>26</td>
</tr>
<tr>
<td>4.5</td>
<td>LZSS pseudocode</td>
<td>27</td>
</tr>
<tr>
<td>4.6</td>
<td>UseCases Diagram</td>
<td>28</td>
</tr>
<tr>
<td>4.7</td>
<td>Session management flowchart</td>
<td>29</td>
</tr>
<tr>
<td>4.8</td>
<td>Classes Diagram</td>
<td>30</td>
</tr>
<tr>
<td>4.9</td>
<td>Block diagram of the overall operation software level</td>
<td>32</td>
</tr>
<tr>
<td>4.10</td>
<td>Reception Sequence Diagram</td>
<td>33</td>
</tr>
<tr>
<td>4.11</td>
<td>Transfer Sequence Diagram</td>
<td>34</td>
</tr>
<tr>
<td>5.1</td>
<td>Compression Time vs Packet Size</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>Decompression Time vs Packet Size</td>
<td>36</td>
</tr>
<tr>
<td>5.3</td>
<td>Compression and decompression time trend</td>
<td>37</td>
</tr>
<tr>
<td>5.4</td>
<td>Compression ratio vs Packet Size</td>
<td>37</td>
</tr>
<tr>
<td>5.5</td>
<td>Execution time vs Number of Packets</td>
<td>38</td>
</tr>
<tr>
<td>5.6</td>
<td>Difference between sender time and receiver time over number of packets</td>
<td>39</td>
</tr>
<tr>
<td>6.1</td>
<td>Initial Gantt Diagram</td>
<td>44</td>
</tr>
</tbody>
</table>
List of Tables

6.1 Planning phase tasks and dedication hours .................................. 40
6.2 Preparation phase tasks and dedication hours ............................... 41
6.3 Implementation phase tasks and dedication hours ........................... 42
6.4 Presentation phase tasks and dedication hours ................................ 42
6.5 Resources list ................................................................................. 43
6.6 Amount of planning hours ............................................................... 43
6.7 Planning phase budget without contingency ...................................... 45
6.8 Preparation phase budget without contingency ................................. 46
6.9 Implementation and testing phase budget without contingency .......... 47
6.10 Total budget with and without contingency ..................................... 47
6.11 Presentation phase budget without contingency .............................. 48
6.12 Sustainability Table ........................................................................ 49
Glossary

Bit  A bit is the basic unit of information in computing and digital communications. A bit can have only one of two values, and may therefore be physically implemented with a two-state device. These values are most commonly represented as either a 0 or 1. The term bit is a portmanteau of binary digit. 15, 31

Byte  The byte is a unit of digital information in computing and telecommunications that most commonly consists of eight bits. 14, 15, 19, 27, 31, 35

Ethernet  Ethernet is a family of computer networking technologies for local area networks and metropolitan area networks. 35

Loopback Interface  Loopback, or loop-back, refers to the routing of electronic signals, digital data streams, or flows of items back to their source without intentional processing or modification. This is primarily a means of testing the transmission or transportation infrastructure. 9, 38

Makefile  Makefile(s) are text files written in a certain prescribed syntax. It helps build a software from its source files, a way to organize code, and its compilation and linking. 10

Node  In communication networks, a node is either a connection point, a redistribution point or a communication endpoint. 21

Payload  Is the cargo of a data transmission. It is the part of the transmitted data which is the fundamental purpose of the transmission, to the exclusion of information sent with it (such as headers or metadata, sometimes referred to as overhead data) solely to facilitate delivery. 13, 35

SCRUM  Is an iterative and incremental agile software development framework for managing product development. 10

Socket  Is an endpoint of an inter-process communication flow across a computer network. Today, most communication between computers is based on the Internet Protocol; therefore most network sockets are Internet sockets. 27

State Matrix  It is a 4x4 bytes matrix which is the input of AES Algorithm. 19, 33

XOR  Exclusive disjunction or exclusive or is a logical operation that outputs true whenever both inputs differ (one is true, the other is false). 24
Acronyms

**AES** Advanced Encryption Standard. 15, 19, 24

**AH** Authentication Header. 13

**CID** Context Identifier. 13

**CPU** Central Processing Unit. 8

**DES** Data Encryption Standard. 15

**ECB** Electronic CodeBook. 15

**ESP** Encapsulating Security Protocol. 13

**FPGA** Field Programmable Gate Array. 9

**GCM** Galois/Counter Mode. 15

**IKE** Internet Key Exchange. 13

**IP** Internet Protocol. 12–14, 18, 31, 35, 50

**IPCom** IP Payload Compression Protocol. 12

**IPSec** Internet Protocol Security. 12, 13

**IPv6** Internet Protocol Version 6. 13

**MTU** Maxim Transfer Unit. 35

**NAT** Network Address Translation. 13

**NIST** National Institute of Standards and Technologies. 25, 35

**OCB** Offset CodeBook. 15, 19

**TCP** Transport Control Protocol. 13
Chapter 1

Introduction

1.1 Origins

Since 1969 J.C.R. Licklider dreamed about the idea of an intergalactic computer network, concept has matured into what we know today as cloud computing. The idea behind this concept is simple: Why not have all the capabilities of a computer but with the advantages of being in the network?; Storage, calculation capacity, running applications or even whole machines instances 24 hours a day and accessible from anywhere in the world.

As a result of this technological improvement, has grown considerably the amount of information circulating daily in our networks, not just the quantity but also the relevance of this, therefore becomes very important the ability to reduce this information (Compression) and make this information can not be read by third parties (Encryption).

To provide the highest possible service for large volume of current information, infrastructures which we know as data processing centers are needed. It is in this infrastructure which will focus our work, specifically on improving the speed and security of virtual machines migrations process between different processing centers.

1.2 Motivation and Objectives

The main idea of this project is a software implementation of a crypto-compressor module for data centers whose design can be implemented on hardware later.

Next to the main objective of the project there are some other objectives, whose naturalness is more didactic, like learn and analyze some compression and encryption algorithms or make software designs for hardware solutions, because the best software solution can be the worst hardware solution or even not implementable.

On the other hand, the main motivation for this project is derived from a short analysis of the Central Processing Unit (CPU) resources that are used to encrypt and compress data. Decoupling these two features, we can reduce the activity of the CPU or use it to process other data, something that is very interesting if you need to move large amounts of information and securely over the network.
1.3 Actors

In this section we analyze who are the principal persons or entities which participate in this project, either directly or indirectly.

1.3.1 Project Director

In this case it is Luis Velasco Esteban, professor in the "Universitat Politecnica de Catalunya". His role in this project is to oversee that the project objectives are adhered to the stipulated time, and guide the student in the different decision-making that arise throughout the project.

1.3.2 Developer

This is the role developed by the student, in this case me. It is the key actor in any informatic project.

1.3.3 Datacenters

Specifically it would be companies offering cloud services that could benefit from the technology developed in this project, since it is these that require transmit large volumes of data securely and quickly.

1.4 Possible obstacles and solutions

1.4.1 Inability to access real datacenter

While the module is designed for operation in datacenters we can not verify that it works in a real one. Despite this handicap, the module will be developed to transmit data between two points and not broadcast. Therefore an environment with two machines and two routers, or even the Loopback Interface, should be enough for valid test both ends, and the proper running of packet forwarding.

1.4.2 Bugs

The obstacle for excellence in every project, so despite not being specific to this project we think is an obstacle to take into account and especially for short-term projects where times must balance perfectly if you want to deliver in the agreed time. Despite being a relatively uncontrollable, we use tools and methods to try to prevent them, as they can be an IDE or unit tests for each project class.

1.4.3 Hardware vs Software

As mentioned above the final idea is to implement this module in a Field Programmable Gate Array (FPGA), therefore we have the problem that it is possible that the solutions adopted for the implementation are not the most optimal from a software point of view, so during the project we should always bear in mind the final hardware implementation. Therefore this will do more difficult and careful the decision making during implementation.
CHAPTER 1. INTRODUCTION

1.5 Working Tools

1.5.1 Git

In any project in which code is developed incrementally, it is important to have a version control. In this case we will use git as version control, inasmuch as is which commonly use in other projects and my knowledge on it is more extensive than on the other programs of the same nature. Also thanks to its system upgrade project (Commit) all updates will be documented and ordered chronologically, with possibility of returning to previous versions if is necessary.

1.5.2 Netbeans IDE

Although the project may be developed without IDE, we decided use it for several reasons:

• Specific plugin for c++ (language in which the module will be developed) with correction syntax and snippets.

• Integrated support for working with git. Execution of basic commands by graphic interface.

• Ability to self-generate test structures for all kinds of project

• Automatic generation and update of the Makefile.

In short, the IDE is responsible for performing certain tasks that would be routine and repetitive for the programmer, thus saving us time and effort.

1.5.3 Routers

We will use two routers, property of the university, to verify proper operation of the module at the end of its development and during the testing procedure.

1.5.4 PC

The code development and the testing phase will be carry out using the personal computer and the university PC

1.6 Work Method

In this case and after analyzing the different working methods that exist, we think that the best option would be to use an agile method, near to the SCRUM model but with slight variations to adapt it to the specific project needs.

One of the most important changes is the replacement of daily meetings due to the incompatibility of these with attending classes, both tutor and I, for communication via email or any other method currently existing. Additionally, the tutor may at any time review progress on the project, as this will develop into an accessible repository for the tutor and documentation will be added to a shared folder.

For this project, fortnightly meetings will arise for check the right project progress and we will arrange the tasks until the next meeting.
1.7 Validation Method

As part of the development process, test kits will be implemented during the fortnightly meetings to ensure that the objectives agreed at the previous meeting were achieved satisfactorily.

Additionally, we shorten one week the delivery date with the purpose of do some integral module tests.

Thus ensures proper operation during and at the end of this.
Chapter 2

State of the art

In the next section we will detail, point by point, all those determinants concepts for the project and what their state of the art.

2.1 Network

As discussed earlier in this document, this project will provide guidance for further project that aims to add this module within a network card. Therefore it is important to analyze certain existing protocols in reference to data compression on Internet Protocol (IP), as can be IP Payload Compression Protocol (IPCom) [1], or those related to encryption like Internet Protocol Security (IPSec) [12].

2.1.1 IP Compression

Regarding IPCom is important to emphasize certain restrictions or recommendations directly related to the project, as this protocol specify, is necessary to do compression process before fragmentation or encryption of packets. Another important detail that sets us IPCom is the fact that each package must be compressed independently, because the information it contains should be able to be decompressed at the end-point, despite arriving out of order.

For these and other constraints has decided documented on the various existing techniques related to IP Compression [9]:

- **Bulk Compression:** This is the most used technique and involves the use of a shared dictionary between both sides of communication path. This technique, despite obtaining large compression ratios, has the disadvantage that it need to synchronize dictionaries, since all packets communicated over a desynchronization period are discarded by the receiver, causing the issuer must forward these.

- **Packet by Packet Dictionary:** This technique is the other end of the matter, here each packet is compressed individually and whole context is included in the package. Thus the need for synchronization is avoided, since each packet contains information to be decompressed, and the level of memory used in the system is minimal. On the downside, this technique obtain lower compression ratios than dictionary technique.
• **Guess-Table-Based compression:** This technique is based on the next bytes prediction of the message, based on the bytes above. If the prediction is correct, these bytes are added to a table identical for the transmitter and receiver, and so sending these through the network is not necessary.

All techniques listed above are used for compressing the data itself, the Payload of each packet. Therefore the question of whether there was a possibility of compressing the headers of each package, thereby increasing the compression ratio was raised. Here we will discuss some of the techniques used for this purpose:

• **Van Jacobson:** The idea behind this technique is generate a Context Identifier (CID) which are identical for certain header fields are alike. For example two packages with the same IP addresses, the same IP protocol and the same Transport Control Protocol (TCP) ports could share the same CID. This is a considerable reduction in the size of the header as such in this case can convert multiple bytes in one, with which could generate up to 512 CID. The problem with this kind of compression is the incompatibility with several technologies recently added as Internet Protocol Version 6 (IPv6) among others.

• **IP:** Is based on the same principles as Van Jacobsen compression, find equalities between packets that can be omitted, the only difference is that this does not drop the IP level to find those similarities. The advantages of this technique lies in the independence of the transport protocol but loses effectiveness against Van Jacobsen when the transport protocol is TCP.

I will like close this section saying that the header compression will not be implemented in final design, due to some technical implications that we go to explain in chapter 3.

### 2.1.2 IP Encryption

IPSec is a set of protocols designed to provide secure IP communications, encrypting packets during communication. This standard are formed by two security protocols at packet level and one designed for secure key exchange [3]:

• **Authentication Header (AH):** It provides authentication, integrity and non-repudiation creating a unique label for each package, using a secret key and immutable parts of the package. Operates a level above IP.

• **Encapsulating Security Protocol (ESP):** It provides confidentiality and optionally, authentication and integrity protection. This protocol works at IP level, so despite not being able to protect the IP headers, it can be used over Network Address Translation (NAT), main difference from AH.

• **Internet Key Exchange (IKE):** This protocol allows the exchange of secret keys required for encryption process. To carry out the exchange uses the Diffie-Hellman protocol that we can see in figure 3.5.
2.2 Compression

In this section we will focus on existing concrete algorithm, which can be applied to the IP compression techniques mentioned in the previous paragraph.

To speak of compression algorithms is important to note that any existing algorithm can be classified into two major groups, lossless compression algorithm and lossy compression algorithms. The latter will not be discussed in this paper because for the goal that concerns us can not be applied, because we must preserve information completely.

Focusing our attention on the lossless algorithms we find many of these, but above all there are 3 families of algorithms [11] from which derive the majority, and we will detail below:

- **Run-Length Encoding**: This is the simplest existing compression algorithm. Lie in group consecutive Bytes that are identical and format the output, so that we have the specific Byte and the number of times it is repeated. It is understandable that in a single Byte can reference up to 512 values, so the compression level can be very high if these characters are repeated consecutively.

- **Huffman**: It is one of the most classical compression algorithm. Described by David A. Huffman in 1952 in *A Method for the Construction of Minimum-Redundancy Codes* [5], this algorithm takes an alphabet of n symbols and occurrence frequencies of the data to form a Huffman code for this particular information. It uses binary trees as a data structure. The basic idea of this algorithm is reencode the most commonly used symbols to occupy the least space possible bitwise, therefore reducing the size of the most frequent bytes can be achieved a great reduction in information total size.

- **Lempel-Ziv**: This compression algorithm is the most used, particularly variations that provide some improvement such as LZ78, LZW or LZSS [4]. Unlike Huffman that requires going through the information twice, once to find the frequencies and another to compress, Lempel-Ziv compress while it moves through the information. It uses a concept known as sliding window, that is a structure where a portion of the previous treated information is stored. To achieve compress the information it uses tuples \{length, size\} which define us how many bytes must fall back on the sliding window and how many bytes must be copied to the output buffer, considering that the current byte in all probability have appeared previously. Therefore there are two factors that can determine the efficiency of this family of algorithms, first the size of the sliding window, with larger size we can obtain better compression but we penalized in the time it takes to do so. On the other hand the size in bytes of the tuples is vital, since an excessively small size will lose the ability to reference more data, thus penalizing the compression ratio, but it is too big we meet with the handicap of time increment. Therefore calibrate these two factors is vital for optimal results with this kind of algorithms.

Finally, for our module we will use LZSS, because it have fast hardware implementations and offer more flexibility due to its parameters configuration. In 4.2 section we will see how works the algorithm and a little explanation of how we implemented.
2.3 Encryption

In case of encryption exists two possible divisions for kind of data treatment, and another two for kind of key.

On one side of data treatment, we have flow algorithms, which perform encryption at Byte-level or even Bit-level. Usually encrypt process takes less time. Some of the more current ones are RC4 or Rabbit. In the other side, we find the block algorithms, such as its name suggests, groups of bytes are formed and certain predefined transformations are applied on these, according to the algorithm, over a defined number of rounds. These algorithms are Data Encryption Standard (DES), Advanced Encryption Standard (AES) and others.

Taking into account the algorithm key we have symmetric algorithm, which uses same key to encrypt and decrypt, it is the case of AES, and asymmetric algorithm which uses a couple of keys formed for a public key, used to encrypt, and a private key used to decrypt.

While the task to find the ideal algorithm for our project could be overwhelming by the amount of options we can find, finally we decided focus our implementation on AES, due to its time/security ratio. We found faster algorithm than AES, like Blowfish for example, but for this faster algorithm exists attacks that put in risk its security. Another reason to choose AES was the possibility to implement it on hardware.

Therefore, focusing more about AES [6] [7] found that there are uncountable versions of this algorithm. We find versions for different key sizes (128, 192 and 256 bits), and versions with different operation modes as Galois/Counter Mode (GCM), Electronic CodeBook (ECB) and Offset CodeBook (OCB). The key size offer more security to the algorithm for larger sizes, but increase process time. The operation modes add to the main algorithm some interesting characteristics, such a confidentiality or authenticity, through a tag generation of the input data.

For our project we will use AES [14] with 256-bit key, without any mode of operation, for performance reasons that we will explain in 3.1.2 section.

Finally our implementation of encryption module is based on the presented in "Design and implementation of a hybrid encryption scheme based on DHIES" [2]. On 3.1.3 section of this document, the structure of our module will be explained and the need for its hybrid design, concept that will become clear in the same section.

2.4 Unifying Concepts

Finally and as closure, it would be advisable to analyze the existence of the two project concepts of greater importance, but in a unified way [15], ie if there is a module or program that uses compression and encryption simultaneously.

After an exhaustive search, the greatest similarity with the concept that we want to apply in this work have been different compressors which facilitate the use of encryption via parameters.
The failure to find the same or similar product to which we want to developed is more than possible cause that the compression and encryption are completely independent and therefore not require an interaction between them.
Chapter 3

Hardware Design

We propose a hardware implementation in general terms, without going into the small details, but if suggesting some improvements over the basic design that could be used in future design.

3.1 Proposed Implementation

In this section we go to explain how we did the module segmentation design and we expose some possible improvements. All the possible improvements mentioned in this section have not been implemented in the software design because they were not our goal.

3.1.1 Compression Module

This module it is formed by two different big parts, the compressor and the decompressor. These two parts have their own input and output buffer, for this reason the compression module design have two inputs and two outputs. The main idea over use two different buffers is allow the parallel compression and decompression without the need to worry if the input data should be compressed or decompressed.

![Compressor basic design](image)

Figure 3.1: Compressor basic design
The implemented algorithm for our compression module is a packet by packet LZSS algorithm, which is explained on 4.2 section. These kind of implementation offers us the possibility of reply the compressor and the decompressor many times how we want, inasmuch as every chunk\(^1\) of information is treated independently and do not have any kind of dependence with other chunks. The only consideration to be taken into account when one of this components is replicated is the need to add two more buffers per replica to design, due to buffers have not idea on how to distinguish every chunk, inasmuch as this buffers works on byte level and chunks can have different sizes.

\(^1\)In compression module we consider a chunk of data the payload extracted from an IP Packet.
3.1.2 Encryption Module

Despite at first time this module would implement AES256-OCB algorithm, finally we decided implement AES256 algorithm without OCB operation due to our poor compression ratio, which we can see in figure 5.4, inasmuch as operation modes add additional Bytes to the information that we could not assume because our packet size will be greater than the packet size without treatment.

The main idea in this hardware design is the same exposed on compression module design. We have two big components, encryptor and decryptor, every component have their own input and output buffer, but in this case, the module only have one buffer for input and another for output since the input chunks\(^2\) only have order dependency and, in contrast with compressor module buffers, in this case we know chunk size previously, concretely 16 bytes, that is the State Matrix size.

![Figure 3.3: Encryptor Basic design](image)

When we talk about order dependency, we are trying to say that encrypt one chunk in a different encryptor of the previous chunk doesn’t affect the output, because the process of compression and decompression of every chunk is independent provided that order is maintained in the output, ie the block that entered first into module, will be the first to leave, being that not respect the order, information is completely disordered, losing its integrity.

AES offers an interesting characteristic for hardware implementation. The algorithm is divided in four byte level transformation, explained in 4.1 section, which are applied in a determinate number of rounds, for this reason, exist the possibility of segmentation this functions and apply pipelining over the functions. The figure 3.6 explain the idea in a better way.

Finally we go to explain the reasons for do our design hybrid. First of all, the concept hybrid in encrypt modules refers to how the key is exchanged. How we explained in 2.3 section, exist two kind of encryption algorithm, symmetric and asymmetric algorithm. The idea in hybrid modules is take advantage of both paradigms. We can use a

\(^2\)In Encryption Module we consider a chunk of data a block of 16 consecutive bytes
symmetric algorithm, which ones have better performance and better security, without
sending the symmetric key over the network inasmuch as the endpoints can calculate
the symmetric key using the public and private key generated by the asymmetric algo-

rithm, as we can see in figure 3.5

In order to understand the figure properly we are going to explain an example:

1. Alice and Bob agree to use a prime number $p = 23$ and base $g = 5$ (which is a
   primitive root modulo 23)

2. Alice chooses a secret integer $a = 6$, then sends Bob $A = g^a \mod p$
   - $A = 5^6 \mod 23 = 8$

3. Bob chooses a secret integer $b = 15$, then sends Alice $B = g^b \mod p$
   - $B = 5^{15} \mod 23 = 19$

4. Alice computes $s = B^a \mod p$
   - $s = 19^6 \mod 23 = 2$

5. Bob computes $s = A^b \mod p$
   - $s = 8^{15} \mod 23 = 2$

6. Alice and Bob now share a secret (the number 2)
3.1.3 Network Module

This module is responsible of 3 important processes:

- Detect incoming packets over the established session between the transfer points
- Filter packets to be processed
- Send the processed packets to the network

We can find our inspiration in the FPGA network cards implementation, like in Design and Evaluation of FPGA-Based Gigabit-Ethernet/PCI Network Interface Card [8], the only difference is apply filters over some bits of packet header in order to process only packets that should be processed.

In order to implement this filter easier, the standard [13] offers a range of values (253-254) in protocol header field for experimental and testing purpose. We can use the 253 protocol to recognize packets to be processed.

To end this section I would like to discuss the main reason that the compression is not performed on the headers, this lies in the fact that when making connections through the network, we can not ensure that the intermediate Nodes know how to interpret compression, thus may lose all packages.
Figure 3.6: Pipelining over AES transformation functions
3.2 Unified Design

In this section we go to see the complete module block diagram and we will do some comments about it.

We can see that we have two possible paths in our design. The blue path it’s applied over the outbound packets and the green path to the incoming.

It is important pay attention in the fact that the both paths are the inverse of each other, the reason seems obvious, since the application order of the operations (compression and encryption) affects the final result. In our implementation we decide apply first the compression because the encryption affects the variability of the input bytes and could make lose a lot of compression ratio due to this variability.
Chapter 4

Software Design

In this section software solution proposed will be discussed. First of all, we will detail different classes that form our implementation, from most low level classes to higher level classes, accompanied by some diagrams, to help reader assimilate all information.

Our Software implementation is the most similar to the hardware implementation to provide a solid reference for the future FPGA implementation. Despite those similarities we introduced some software singularities, such as the definition of some structures that make our software implementation easier without affecting the possible hardware implementation, and explained in detail in 4.4 section, because it is the class affected by those singularities.

4.1 AES

AES algorithm does not present much possibilities in its implementation due to its clearly defined structure, which we can see in figure 4.1. For our case, we use a 256 bits key, so AES specification says that the number of rounds for this length is 14, it is unique change with respect to other key lengths. Therefore the structure remains equal, for this reason lets go to explain the operations that form the algorithm and the relevant implemented features and possible features.

4.1.1 AddRoundKeys

In the addRoundKey step, a portion of the key, named subkey, is added to the state matrix by a XOR operation. Subkey is derived from the main key using Rinjdael’s Key Schedule, which pseudocode is in figure 4.2.

At this step any relevant improvement was made, because it is only a XOR operation.

4.1.2 SubBytes

The SubBytes transformation is a non-linear byte substitution that operates independently on each byte of the State Matrix using a substitution table (S-box) defined in figure 4.3.

The S-box table is obtained by applying some transforms to a finite field, but this
On this step, first of all, we decided calculate the values in order to do not spend memory, because in FPGA the memory is a valuable resource. But according as the project progress, we see that our memory outlay was low, for this reason, finally we implemented this operation by a matrix stored in memory.

![Figure 4.1: AES algorithm structure](image1)

![Figure 4.2: Rinjdael’s key schedule pseudocode](image2)
4.1.3 ShiftRows

This is the easiest step, the bytes on the last three rows of the state matrix are shifted many times how its row number indicate, taking into account that first row index is 0. In figure 4.4 we have graphical process explanation.

This step does not offer any good improvement because is so easy implement it on hardware.

4.1.4 MixColumns

Is the same idea that in SubBytes step, but now we applied different operations in the same finite field, but per columns instead of per byte. All mathematical information is on [10].

As in SubByte step the same memory issue appear. In this case we chose calculate all values instead of memorize them, because to do the encryption and decryption we need and amount of 6 table which size is 256 bytes each table, 1.5MB in total.

Figure 4.3: S-box: substitution values for the byte xy (in hexadecimal format)

Figure 4.4: ShiftRows operation
4.2 LZSS

As we have explained the main idea of how the algorithm works in the state of the art section, we go to see how it works deeply, and all the improvements that we add on our implementation.

First of all we go to remember all those important concepts to understand the algorithm. Sliding window is one of them, and it is a buffer that stored a portion of read data. The size of this portion is marked by the number of bits that we use in our keywords.

Keyword is another important concept of this algorithm, it is a tuple (length, size). Length defines our initial position when we are searching in sliding windows, and size how long could be the string. In our implementation keywords are of 16 bits, so if we find more than 2 Bytes repeated string that we could compress, we obtain compression.

In figure 4.5, we can see how the algorithm works in general. The main idea is simple, it search at input buffer strings stored in the sliding windows, if it find a repeated string larger than the keyword size, in our case 2 Bytes, instead of put all the string on output buffer, it puts a keyword. In that figure, we can see another important concept of the LZSS algorithm, the flags buffer. Algorithm put a 0 when it introduce a byte in the output buffer, and put a 1 when it introduce a keyword on the output.

```plaintext
while (lookAheadBuffer not empty)
{
    get a pointer(position, match) to the longest match in the window for the lookAhead buffer;
    if (length > MINIMUM_MATCH_LENGTH)
    {
        output a (position, length) pair;
        put 0 into flags buffer
        shift the window length characters along;
    }
    else
    {
        output the first character in the lookAhead buffer;
        put 1 into flags buffer
        shift the window 1 character along;
    }
}
```

Figure 4.5: LZSS pseudocode

On flags buffer we introduced an implementation feature, we made this buffer circular. Instead of shifting the sliding window length characters when a match is found (which is too slow), we just insert to the sliding window position mod N whenever we need to write the new characters, where N is the size of the sliding window. This circular access technique already provides a great speed-up to the basic algorithm, and by creating a better search function, the program can perform more efficiently.

4.3 Network Manager

As all implementations with network activity, we need that a part of our program manage our connections. This class is responsible of create and destroy Sockets, read from and write into network buffers and filter packets that must not be processed.
CHAPTER 4. SOFTWARE DESIGN

As we exposed in 3.1.3 section, our filter checks protocol field of packet header, and if its value is 253 put the packet in the input buffer to be processed.

4.4 Core Manager

This class is the main controller of our module. As we can see in figure 4.8 all rest of the classes are connected to them. This means that our Core Manager is formed by a Network Manager, a LZSS, an AES, and a buffer controller, which is explained latter on this section.

In figure 4.9 we can see how it works. Basically it is the director, which knows the state of all other components, and be in charge of all the other components work properly. As a controller, this class is the only one that have operations that user can call, however the set of possible operations is so short due to this module must work like an automaton, the user only have to start the connection and select in which mode want to operate, in client mode or in server mode. As we can see in figure 4.6, the most important operations of our module are transfer and reception, transfer corresponds to server mode and reception to client mode.

![Figure 4.6: UseCases Diagram](image)

Transfer operation consist of a group of functions which are part of the other classes that form our module. In figure 4.11 we have a sequence diagram that explain this process. As we can see the operation change the state of the module to in transfer, and read all packets that arrive to its interface applying the next process for each packet:

- 1.4 - Put all packet bytes into our input compression buffer
- 1.6 - Compress data
• 1.8 - Take the output compression buffer, and group the output bytes into 16 bytes matrix, and put these matrix into the encrypt input buffer

• 1.10 - Encrypt data

• 1.12 - Put all 16 bytes matrix of encrypt output buffer as individual bytes, without format

• 1.14 - Send packet to network manager, which sen over the network

The reception process, which we can see in figure 4.10, is the same process but reverting the operations.

Finally, how we commented at the start of this chapter, we introduced some relevant structures to this class that we explain below.

Figure 4.7: Session management flowchart
4.4.1 Session

For software implementation we have created a structure that we call session and these structures are used by the module to manage connections between hosts and most importantly, passwords of the encryption algorithm used at any particular time. These structures are formed by a series of fields:

- **ID**: Define a unique identifier for each connection
- **Live Time**: Define the duration of our session. This time is renewed each time that connection is used, so the session expiration is given by long periods of disuse.
- **Connection**: It is another complex structure which is formed by different fields that you can see in figure 4.8
- **Source Address**: IP source address of connection
- **Destination Address**: IP destination address of connection

The key requested during the process is obtained from a key manager, which handles the process seen in figure 4.7, which is not implemented on this project because is out of our goals. With purpose of do some test we implemented a stub that always return the same key.

4.4.2 Buffers Controller

In our Software design, we are define 3 kinds of buffers with different characteristics in their structure which ones offers us an easiest way for process all the data. This class is responsible for managing these buffers that are explained below.

**IPBuffer**

This kind of buffer is responsible for storing IP messages, which, in our implementation, are represented by the IPMessage structure, visible in Figure 4.8. These buffers are present at the input and output of our module and are useful because they facilitate processes such as separating the header or differentiation between different packages, more difficult process if the buffer only understand packets at Byte or Bit level.

**RawBuffer**

These buffer stores pure bytes and is responsible for sending the information to the compressor and the decompressor. This buffer is the most similar to what we would find in a hardware implementation, but by working at the byte level for a software implementation would require an additional effort in controlling the actual context of information, i.e. is where it starts and ends a block that shares information context.
Figure 4.9: Block diagram of the overall operation software level
AESBuffer

These are the buffers responsible for managing the input and output of our encrypting module. These buffers store optimally formatted information to be processed by the module. Lists of 16 bytes matrix are stored, thus speaking from the viewpoint of the algorithm State Matrix are stored.

![Figure 4.10: Reception Sequence Diagram](image)

Figure 4.10: Reception Sequence Diagram
Figure 4.11: Transfer Sequence Diagram
Chapter 5

Software Evaluation

For testing performance of our module have been taken into consideration various aspects that we discuss below.

First of all we would like to comment that both compression and encryption modules are individually tested in unit testing. In the encryption module have been used official test vectors by NIST obtained in "The Advanced Encryption Standard Algorithm Validation Suite (AESAVS)". For the compressor module, tests have been performed on data sizes 8 to 1500 Bytes.

We have chosen the maximum size 1500 Bytes despite IP supports 65,536 Bytes of datagram size, because the vast majority of networks operate on the Ethernet protocol at network level, whose Maxim Transfer Unit (MTU) is 1500 Bytes, so testing with high values inputs not provide relevant information to our project.

Therefore, the load test conducted at the end of this section also will be made with packet size 500 for the Payload.

5.1 Compression Unit Test

In the next section we will see some graphs relating the rate of compression/decompression in function of the variability of the input, and the times in which this is done. The variability of input characters is simulated by creating multiple random entries in which the set of possible characters increases by 1 at each iteration, i.e. in the first iteration the message will consist of a single character, the following may be formed up to 2 different characters and so on.

5.1.1 Times

As we can see in figures 5.1 and 5.2, the compression time is really higher than the decompression time. One of the possible reasons is the variability of the input, because these test are done with the maximum variability in a Byte, 128 possible values. This fact affects directly the compression ratio, how we could see in the next section,causing a compression ratio lower than 1, so if the compression ratio is lower than 1, this means that the decompressor does not perform any task, explaining these low times.
Analyzing the trend of the times we can extract another interesting conclusion, with the same variability, larger sizes affects more the compression process than the decompression process, so the decompression process is more scalable than compression in our implementation.
Figure 5.3: Compression and decompression time trend

5.1.2 Compression Ratio

How we can see in the figure 5.4, we achieve a better compression ratio with low variability and higher sizes. In this case the results were expected, since it seems obvious to think that with less variability increases the likelihood of recurrence of strings, and with larger sizes, increases the probability of finding them.

Figure 5.4: Compression ratio vs Packet Size

Another important aspect to bear in mind is the absolute value of compression ratio, inasmuch as it is so poor for high variability and high sizes, that is the most common of our possible scenarios. One of the possible reasons falls in the fact that the maximum value of our input is so small for our algorithm parameters or even for these kind of algorithm
5.2 Load Test

The purpose of this section is to analyze the time when the complete communication between client and server is performed, i.e. the time that the server takes to compress-encrypted-send data, and the server takes to receive-decrypt-decompress these for an increasing number of packets. The time of client and server will be taken individually to analyze possible deviations in the time between the client and server.

To perform these tests has finally used the Loopback Interface, so the communication times should not unduly affecting the total process time.

![Figure 5.5: Execution time vs Number of Packets](image)

The results are as expected. The difference between client and server time is minimum as shown in figure 5.5 and the execution time increase linearly in reference with the number of packets (5.6). Another interesting information that we could extract of the results is the throughput of our module with this test conditions. We send packets of 500 bytes, so our throughput is about 0.42 MB/s. It is under current connections throughput offered by datacenters, but we must remember that the final design will be implemented on hardware, which probably offer more design options and better performance.
Figure 5.6: Difference between sender time and receiver time over number of packets
Chapter 6

Project Management

6.1 Temporal Planning

This project intends to be developed in about five months, with the deadline established on January 26, 2015, when the project aims to be presented in court.

In next sections we explain how is divided our ideal temporal planning, and in final of each division, you can find some comments about digressions suffer in fact. You can see a general overview of the temporal planning in Gantt in figure 6.1 and in temporal planning summary in figure 6.6

6.1.1 Planning

This is the initial phase of any project which has a stipulated duration of one month.

Most of this project phase focuses on the GEP course and consists of several parts or "tasks" that help better initial organization.

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope definition</td>
<td>9,25</td>
</tr>
<tr>
<td>Temporal planning</td>
<td>8,25</td>
</tr>
<tr>
<td>Economic management and sustainability</td>
<td>9,25</td>
</tr>
<tr>
<td>Preliminary presentation</td>
<td>6,25</td>
</tr>
<tr>
<td>Context and biography</td>
<td>15,25</td>
</tr>
<tr>
<td>Specialty</td>
<td>10</td>
</tr>
<tr>
<td>Initial milestone presentation</td>
<td>18,25</td>
</tr>
</tbody>
</table>

Table 6.1: Planning phase tasks and dedication hours

Adding hours to carry out the planning phase we have a total of 76.5 hours to distribute within 1 month of work, assuming that would not work on weekends we get a working ratio of 3.06 hours per day of dedication.

This phase has been followed from beginning to the end, without any digression, as the scheduling of this phase was imposed by the university.
6.1.2 Preparation

In this phase of the project will be carried out all tasks related to satisfaction of some existing needs to work on implementing ongoing after phase start.

Therefore in this part of the project we find everything related to project’s elements configuration and acquiring the necessary software and hardware, like we could see in figure 6.2

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration of documentation environment</td>
<td>1.25</td>
</tr>
<tr>
<td>Software Acquisition</td>
<td>4</td>
</tr>
<tr>
<td>Configuration of deployment environment (software)</td>
<td>8</td>
</tr>
<tr>
<td>Hardware Acquisition</td>
<td>3</td>
</tr>
<tr>
<td>Configuration of deployment environment (hardware)</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 6.2: Preparation phase tasks and dedication hours

To carry out the preparation phase an amount of 31.5 hours are needed, assigning a maximum of one week and a half for this phase, we would have a ratio of 4.5 work hours per day, is somewhat higher than in the previous phase, but given the short duration of the project and being wary of possible problems, is an acceptable work ratio for one week and a half.

This phase was almost ineffective due to most of the work that we had to do it was done before this phase or we did it in less time than expected. For this reason, the implementation and testing phase started earlier.

6.1.3 Implementation and testing

This is the phase with a higher weight of the project as it includes the 3 pillars of the project:

- Deployment
- Validation
- Documentation

On this stage a cyclic method of 15 days will be applied in which tasks in figure 6.3 will be performed.
### CHAPTER 6. PROJECT MANAGEMENT

<table>
<thead>
<tr>
<th>Task</th>
<th>Observation/Description</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planification</td>
<td>The cycle objectives will be treated with the tutor</td>
<td>0.5</td>
</tr>
<tr>
<td>Analysis and Organization</td>
<td>The priority and dedication of every objective will be evaluated to complete each of them</td>
<td>2</td>
</tr>
<tr>
<td>Deployment</td>
<td>Implementation of each assigned module</td>
<td>18</td>
</tr>
<tr>
<td>Validation</td>
<td>The right working of each module will be checked</td>
<td>18</td>
</tr>
<tr>
<td>Documentation</td>
<td>Every making decision in deployment and testing phase will be documented</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 6.3: Implementation phase tasks and dedication hours

Adding the hours dedicated, a total of 50.5 hours every 15 days, so it is a ratio of 5.05 hours a day if we have not in mind the weekend. It is a high ratio and possibly difficult to perform in the 2 months planned for this phase, but as mentioned above is an ideal project planning approach.

As we could suspect, this phase suffer some modifications in cycle length. Despite of in our first idea, cycles were formed by 15 days, finally were of 1 month due to some problems to find time for meetings.

#### 6.1.4 Presentation

This is the last phase of our project and will take approximately 2 weeks. This phase consists of a total of 30 hours, thereby obtaining a ratio of 3 hours daily work. We can see task breakdown of this phase in figure 6.4

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation analysis</td>
<td>10</td>
</tr>
<tr>
<td>Presentation support elaboration</td>
<td>15</td>
</tr>
<tr>
<td>Presentation training</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 6.4: Presentation phase tasks and dedication hours

Finally this phase was shorten one week, because coincide with exams period, and I needed time to study.

#### 6.1.5 Resources

Figure 6.5 is a detailed list of human and material resources that we hope use in our project

It is noted that is a resource estimation and therefore the possibility that any material not registered in this list can be used exists, due to, for example, some complications at implementation phase.
6.1.6 Assessing alternatives and action plan

In this section we will make a short analysis of total project time, paying special attention to the most critical tasks and as a delay in these can affect us in project’s planning.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Total Hours</th>
<th>Duration(in working days)</th>
<th>Hours/Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>76,5</td>
<td>25</td>
<td>3,06</td>
</tr>
<tr>
<td>Preparation</td>
<td>31,5</td>
<td>7</td>
<td>4,5</td>
</tr>
<tr>
<td>Implementation and testing</td>
<td>303</td>
<td>60</td>
<td>5,05</td>
</tr>
<tr>
<td>Presentation</td>
<td>30</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>441</strong></td>
<td><strong>102</strong></td>
<td><strong>4,32</strong></td>
</tr>
</tbody>
</table>

As we can see in figure 6.6, the most critical phase of our project is the implementation and testing phase, this being about 70 % of project weight hours, and its hours/day ratio is 21 % higher than the average of the project.

Therefore in case of suffering delays at this stage the only solutions are:

- Recovering lost hours working on weekend
- Increase the hours/day ratio.

All resources that will be used in this project, will be available all days of the week from 8:00 to 20:00, so material availability should not cause any additional delay.

Finally we did not need apply any of these solutions, inasmuch we finished implementation and testing phase in the same number of days but in less hours, specifically in 220 hours.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 PLANIFICACION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Planificación del alcance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Planificación temporal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Gestion Economica y Sostenibilidad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Presentación Preliminar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Contextualización y biografía</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 Especialidad Presentacion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 PREPARACION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Configuración entorno documentación</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 Adquisición software</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 Configuración entorno desarrollo (software)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 Adquisición hardware</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 Configuración entorno desarrollo (hardware)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 IMPLEMENTACION Y TESTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 Planificación, análisis y organización</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 Desarrollo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 Validación</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 Documentación</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 Repetición ciclo anterior</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 PRESENTACION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 Análisis documentación</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22 Elaboración soporte presentación</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23 Práctica presentación</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.1: Initial Gantt Diagram
6.2 Economic Planning

In this section an economic breakdown will be shown in tables of different project phases in relation with the resources of our project. All digressions suffer in fact in relation to the planned budget will be commented at final of pertinent section.

At the end of this section you will find the total project budget considering contingency.

6.2.1 Planning

The budget described in 6.7 belongs to the phase described in figure 6.1. Some minor resources that are not appear in figure 6.5 are added.

This phase, project director is the major fraction of our budget, so we tried to reduce his hours, but was impossible due to his importance. Word problems appeared like we predicted, so this budget did not suffer any modification.

<table>
<thead>
<tr>
<th>Direct Costs Resource</th>
<th>Quantity</th>
<th>Unit price</th>
<th>Time</th>
<th>% amortization</th>
<th>Total</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Director</td>
<td>-</td>
<td>25€/h</td>
<td>66.5h</td>
<td>-</td>
<td>1912.5€</td>
<td>-</td>
</tr>
<tr>
<td>Analyst</td>
<td>-</td>
<td>21.8€/h</td>
<td>10h</td>
<td>-</td>
<td>218€</td>
<td>-</td>
</tr>
<tr>
<td>Macbook Pro</td>
<td>1</td>
<td>1329</td>
<td>-</td>
<td>0.02</td>
<td>26.58€</td>
<td>Util life 48 months, 1 month in this phase</td>
</tr>
<tr>
<td>Office 365</td>
<td>1</td>
<td>7€/month</td>
<td>1 month</td>
<td>-</td>
<td>7€</td>
<td>To write documentation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Costs Resource</th>
<th>Quantity</th>
<th>Price</th>
<th>Time</th>
<th>% amortization</th>
<th>Total</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light</td>
<td>0.32Kw/h</td>
<td>0.126€/Kw</td>
<td>76.5</td>
<td>-</td>
<td>3.08€</td>
<td>300w mac 20w light</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Incidents Concept</th>
<th>Probability of occurrence</th>
<th>Additional Cost</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word difficulties</td>
<td>20%</td>
<td>32.5€</td>
<td>P of 20% over 25€/h in 66.5h</td>
</tr>
</tbody>
</table>

Total in planning phase(without contingency) 2499.66 euros

Table 6.7: Planning phase budget without contingency

6.2.2 Preparation

The budget described in 6.8 belongs to the phase described in figure 6.2. Some minor resources that are not appear in figure 6.5 are added.

Some interesting comment about this budget is use of free software in order to reduce final amount. For our purpose, as much Netbeans IDE as Git offers us all necessary requirements to develop our project with same efficiency and quality that it can offer any proprietary software.

This budget suffer some modifications due to we achieved reduce some software and hardware configuration times, and we had no configuration problems.
Table 6.8: Preparation phase budget without contingency

6.2.3 Implementation and Testing

The budget described in 6.9 belongs to the phase described in figure 6.3. At first we thought use programmer to do tests, but we found that test programmers are cheaper than programmers, for these reason we decided include test programmer to this budget.

We had a lot of problems with implementation and testing, so the additional cost derived of these problems was totally necessary. Despite that, how we commented before, we achieved reduce phase total time, for this reason, final cost is lower than expected.

6.2.4 Presentation

The budget described in 6.11 belongs to the phase described in figure 6.4.

It is the phase that it used less resources but is not the most cheaper, because we use our more expensive resource per hour, project director.

At time to write this memory, we don’t how could be final budget of this phase because we have not entered into this phase yet.
CHAPTER 6. PROJECT MANAGEMENT

### Direct Costs

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quantity</th>
<th>Unit price</th>
<th>Time</th>
<th>% amortization</th>
<th>Total</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Director</td>
<td>-</td>
<td>25€/h</td>
<td>15h</td>
<td></td>
<td>375€</td>
<td></td>
</tr>
<tr>
<td>Programmer</td>
<td>-</td>
<td>18€/h</td>
<td>144h</td>
<td></td>
<td>2592€</td>
<td></td>
</tr>
<tr>
<td>Test Programmer</td>
<td>-</td>
<td>16€/h</td>
<td>144h</td>
<td></td>
<td>2304€</td>
<td></td>
</tr>
<tr>
<td>Macbook Pro</td>
<td>1</td>
<td>1329€</td>
<td>-</td>
<td>0.0625</td>
<td>83.06€</td>
<td>Util life 48 months, 3 months in this phase</td>
</tr>
<tr>
<td>Office 365</td>
<td>1</td>
<td>7€/month</td>
<td>3m</td>
<td></td>
<td>21€</td>
<td>Write Documentation</td>
</tr>
<tr>
<td>Router Cisco 837</td>
<td>2</td>
<td>45€/month</td>
<td>-</td>
<td>0.125</td>
<td>11.25€</td>
<td>Util life 24 months, 3 months in this phase</td>
</tr>
<tr>
<td>PC</td>
<td>3</td>
<td>350€</td>
<td>-</td>
<td>0.0625</td>
<td>65.625€</td>
<td>Util life 48 months, 3 months in this phase</td>
</tr>
<tr>
<td>NetBeans IDE</td>
<td>1</td>
<td>0€</td>
<td>-</td>
<td>-</td>
<td>0€</td>
<td>Free Software</td>
</tr>
<tr>
<td>Github</td>
<td>1</td>
<td>0€</td>
<td>-</td>
<td>-</td>
<td>0€</td>
<td>Free Account</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quantity</th>
<th>Price</th>
<th>Time</th>
<th>% amortization</th>
<th>Total</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light</td>
<td>0.92Kw/h</td>
<td>0.126€/Kw</td>
<td>303h</td>
<td></td>
<td>35.12€</td>
<td>300w mac, 400w x 3 pc, 120w x 2 routers, 20w light</td>
</tr>
</tbody>
</table>

### Indirect Costs

<table>
<thead>
<tr>
<th>Concept</th>
<th>Probability of occurrence</th>
<th>Additional Cost</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Problems</td>
<td>15%</td>
<td>388.8€</td>
<td>P of 15% over 18€/h in 144 hours of work</td>
</tr>
<tr>
<td>Testing Problems</td>
<td>60%</td>
<td>1382.6€</td>
<td>P of 60% over 16€/h in 144 hours of work</td>
</tr>
</tbody>
</table>

Total in preparation phase (without contingency) 598.53 euros

Table 6.9: Implementation and testing phase budget without contingency

#### 6.2.5 Total Cost

As a section closure, we have our total budget in figure 6.10. As we expected, implementation and testing phase is most of our budget, in particular 64%. However, taking into account, that this phase is the 70% of total time, it means that it is not the most expensive phase in proportion. That is presentation phase, in two weeks we spend almost 10% of our budget.

For all commented below, we can suppose that our real budget will be lower than the expected, but not longer.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>2499.66€</td>
</tr>
<tr>
<td>Preparation</td>
<td>598.53€</td>
</tr>
<tr>
<td>Implementation &amp; Testing</td>
<td>7258.51€</td>
</tr>
<tr>
<td>Presentation</td>
<td>921.49€</td>
</tr>
<tr>
<td>Total without Contingency</td>
<td>11278.18€</td>
</tr>
<tr>
<td>Total with Contingency (10%)</td>
<td>12406.01€</td>
</tr>
</tbody>
</table>

Table 6.10: Total budget with and without contingency
Chapter 6. Project Management

### Direct Costs

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quantity</th>
<th>Unit Price</th>
<th>Time</th>
<th>% Amortization</th>
<th>Total</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Director</td>
<td>-</td>
<td>25€/h</td>
<td>30h</td>
<td>-</td>
<td>750€</td>
<td>-</td>
</tr>
<tr>
<td>Macbook Pro</td>
<td>1</td>
<td>1329€</td>
<td>-</td>
<td>0.01</td>
<td>13.29€</td>
<td>Util life 48 months, 2 weeks in this phase</td>
</tr>
<tr>
<td>Office 365</td>
<td>1</td>
<td>7€/month</td>
<td>1m</td>
<td>-</td>
<td>7€</td>
<td>Make Presentation</td>
</tr>
</tbody>
</table>

### Indirect Costs

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quantity</th>
<th>Price</th>
<th>Time</th>
<th>% Amortization</th>
<th>Total</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light</td>
<td>0.32Kw/h</td>
<td>0.126€/Kw</td>
<td>30h</td>
<td>-</td>
<td>1.20€</td>
<td>300w mac 20w light</td>
</tr>
</tbody>
</table>

#### Incidents

<table>
<thead>
<tr>
<th>Concept</th>
<th>Probability of occurrence</th>
<th>Additional Cost</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Point Problems</td>
<td>20%</td>
<td>150€</td>
<td>P of 20% over 25€/h in 30 hours of work</td>
</tr>
</tbody>
</table>

Total in implementation and testing phase (without contingency) 921.49 euros

Table 6.11: Presentation phase budget without contingency

### 6.3 Sustainability

Currently, sustainability takes a lot of importance in any project. It is important analyze aspects like economic sustainability, social and environment impact, etc.

For this reason, Barcelona School of Informatics, puts at our disposal a grid which offers us a quickly way to analyze quantitatively all those matters associated with sustainability.

As we can see in figure 6.12, our project obtain a mark of 38.6, it is a good mark taking into account that the possible values are compressed between -90 and 60.

Finally, as clarification, we want say that in the sustainability table, a line with "+" sign means something positive for treated aspect, and a line with "-" sign means something negative.
<table>
<thead>
<tr>
<th>Sustainability?</th>
<th>Economic</th>
<th>Social</th>
<th>Environmental</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Viability</td>
<td>Quality of life</td>
<td>Resources analysis</td>
</tr>
<tr>
<td>Planning</td>
<td>+Exist cost evaluation</td>
<td>+No one injured</td>
<td>+Low resource consumption</td>
</tr>
<tr>
<td></td>
<td>-Chance to become cheaper</td>
<td>-There is no real need for the product</td>
<td>+No pollution during realization</td>
</tr>
<tr>
<td></td>
<td>+Collaboration with another project</td>
<td>-No great improvement in the consumer</td>
<td></td>
</tr>
<tr>
<td>Assessment</td>
<td>6.4</td>
<td>5.8</td>
<td>9.6</td>
</tr>
<tr>
<td>Results</td>
<td>Final cost vs Forecast</td>
<td>Social Environment Impact</td>
<td>Resource consumption</td>
</tr>
<tr>
<td></td>
<td>+More cheaper than forecast</td>
<td>Does no affect society in any aspect</td>
<td>+Only electricity to operate</td>
</tr>
<tr>
<td></td>
<td>+Performed in less hours</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assessment</td>
<td>10</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Risk</td>
<td>Adaptation to scenario changes</td>
<td>Social Injuries</td>
<td>Environmental Injuries</td>
</tr>
<tr>
<td></td>
<td>-Reimplementation if the technology changes</td>
<td>Does no affect society in any aspect</td>
<td>+Any environmental injuries</td>
</tr>
<tr>
<td></td>
<td>+ Cheaper reimplementation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assessment</td>
<td>-5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total Assessment</td>
<td>36.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.12: Sustainability Table
Chapter 7

Conclusions

The main objective of this work was to implement successfully a module able to compress and encrypt information, specifically IP packages. Therefore, given the generality of the primary objective we can say that is fulfilled completely.

Despite meeting the target, if it is true that some of the possible improvements that could be applied to the basic design, such as header compression or implement a mode of operation for our encryption algorithm, have been unable to be implemented. In addition, the results of our compression module are so poor and probably we will need reanalyze all compression possibilities in order to find better performance on this aspect. Despite this, whole process to reach these conclusions has been very helpful in my training.

First, an important aspect which I extracted from this work have been able to design software solutions being aware that the ultimate goal is to design them on hardware, and recognize the only software solutions that have been used.

Another important aspect that made me think this work is in reference to design specific purpose modules, because I think that the general purpose hardware has reached, or will reach in the coming years, the maximum performance that can offer, and therefore I think in all aspects of computing, will become important to investigate the creation of specific hardware and how to make this engages and communicates with general purpose hardware so that we can overcome the levels of maximum performance.

Finally, and to finish this job, I would like to comment on the wealth of knowledge that has given me the realization of this project and how important it is to round the formation of all graduates.
Acknowledgements

To complete this project just need to thank all those people who have made this project could end in fruition.

To Luis, my director, for providing me this opportunity, give me confidence and guidelines that have allowed much of our analysis.

To my friends Artem and Navarro, for help me when work is accumulated

My cousin Oscar, to achieve during this Christmas, evade me of the project and I could return with renewed energy.

My girlfriend Gala, for putting up with my temper when things not came out and stay awake at night next to me while working.

And finally my parents, who have taught me the value of hard work and education which helps me to go ahead every day.
Bibliography


