RAQUEL
ANDRADE RAINHO

SpaceSheep: Comunicações de satélite para cenários de agricultura inteligente

SpaceSheep: Satellite communications for smart agriculture scenarios

Universidade de Aveiro
2022

RAQUEL
ANDRADE RAINHO

SpaceSheep: Comunicações de satélite para cenários de agricultura inteligente

SpaceSheep: Satellite communications for smart agriculture scenarios

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor Daniel Nunes Corujo, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática, da Universidade de Aveiro, e do Professor Doutor Pedro Alexandre de Sousa Gonçalves, Professor adjunto da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro.

Dedico este trabalho à minha família e amigos pelo incansável apoio.

o júri / the jury

presidente / president

Professor Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar, Universidade de Aveiro

vogais / examiners committee

Doutor Telemaco Melia
General Manager, Echostar Mobile

Professor Doutor Daniel Nunes Corujo
Professor Auxiliar em Regime Laboral, Universidade de Aveiro

agradecimentos / acknowledgements

Depois de 6 anos de altos e baixos, dou por finalizada uma das etapas mais desafiantes da minha vida. Naturalmente, este percurso não foi feito sozinho e, portanto, aproveito este momento para agradecer a todos os que me foram acompanhando, em especial:

Aos meus orientadores, Daniel Corujo e Pedro Gonçalves, pelo constante apoio que me deram durante o desenvolvimento deste trabalho e por se terem mostrado sempre disponíveis para responder às minhas dúvidas.

À minha família, que me apoiou incansavelmente durante todos estes anos. Aos meus pais, os pilares da minha vida, por todos os ensinamentos e por terem criado a base necessária para o meu crescimento, dando-me sempre liberdade para explorar e apoiando todas as minhas decisões. Um grande obrigado por terem prioritizado o meu bem-estar e o dos manos, deixando de parte muitos fins-de-semana para estarem presentes nas nossas atividades. Aos meus irmãos, Inês e David, por todas as "guerras" e brincadeiras que tornaram os meus dias menos aborrecidos. Um obrigado também pelos momentos mais sérios que passámos e pela constante ajuda que me proporcionam, nomeadamente na revisão deste documento.

Ao Miguel, por ter estado sempre ao meu lado quando mais precisei, motivando-me e dando-me forças para continuar sempre que o percurso se tornava mais difícil. Obrigada por me arrancares sorrisos todos os dias com as tuas piadas secas, por ouvires as minhas queixas e por estares sempre pronto a ajudar-me, principalmente na escrita desta dissertação.

Aos meus amigos, não só por todos os momentos de convívio e diversão, mas também por todos os momentos de partilha e entreajuda, que foram fundamentais ao meu bem-estar e me ajudaram a lidar melhor com o stress universitário. Em particular, agradeço à Inês e à Mariana, as minhas parceiras de "guerra", a quem estou enormemente grata por ter conhecido, por me aturarem e estarem sempre disponíveis para mim. Um agradecimento especial também ao Tomás, por me ter emprestado o seu Raspberry Pi, que foi necessário para o desenvolvimento deste trabalho.

Finally, a note of gratitude to EchoStar Mobile, in particular to Telemaco Melia and Jonathan Smith, for providing the access to their satellite terminal and mobile satellite data services.

Palavras Chave

Agricultura Inteligente, Internet das Coisas, Comunicações de Satélite, Formatos de Serialização

Resumo

A necessidade do aumento de produtividade de atividades diárias tem vindo a contribuir para o desenvolvimento de novos sistemas que consigam otimizar essas tarefas. Dentro do sector agrícola, soluções de IoT têm permitido a monitorização autónoma de plantações e animais, reduzindo o esforço humano e, consequentemente, o custo do produto final. Uma dessas soluções foi desenvolvida no âmbito do projeto SheepIT, um sistema de monitorização animal desenvolvido de forma a remover espécies infestantes em vinhas atráves do controlo do comportamento de rebanhos. Para isso, cada animal está equipado com um dispositivo com sensores e atuadores (collar), que monitoriza e condiciona as suas ações. A informação recolhida por estes dispositivos é enviada periodicamente para um nó agregador (gateway) através de nós fixos espalhados pela área de pasto (beacon), onde é então processada e transferida para uma plataforma computacional remota atráves da Internet. Todavia, estes animais deslocam-se tipicamente por extensas áreas com cobertura de rede terrestre fraca ou inexistente, inibindo o correto funcionamento de tal sistema.

Este trabalho visou mitigar a ausência de cobertura comum em áreas rurais. Para tal, uma interface de comunicações satélite foi integrada no projeto SheepIT e, consequentemente, as mensagens trocadas pelo sistema foram adaptadas e otimizadas de forma a responder às limitações desta nova tecnologia. Estas modificações extendem o projeto SheepIT para operar em cenários em que a cobertura de rede terrestre não está disponível.

Keywords Smart Agriculture, Internet of Things, Satellite Communications, Serialization Formats

Abstract

The need to increase productivity in daily activities has been contributing to the development of new systems that can optimize those tasks. Within the agricultural sector, IoT solutions are allowing the autonomous monitoring of crops and animals, reducing human effort and, consequently, the cost of the final product. One of those solutions was developed under the scope of the SheepIT project, which is an animal monitoring system developed to remove weeds in vineyards by controlling the behaviour of herds. To do so, each animal is equiped with a sensor- and actuator-based device (collar), which monitors and conditions its actions. The information these devices collect is periodically forwarded to an aggregator node (gateway) through fixed nodes spread around the pasture area (beacon), where is then processed and uploaded to a remote computational platform via the Internet. However, these animals typically move around extensive areas with poor or non-existent ground network coverage, which inhibits the proper communications operation of such system.

This work aimed to mitigate the common lack of coverage in rural areas. To do so, a satellite communications interface was integrated into the SheepIT project and, consequently, the messages exchanged by the system were adapted and optimized to meet the constraints of this new technology. These modifications extend the SheepIT project to be able to operate in scenarios where ground network coverage is not available.

Contents

List of Figures

List of Tables

List of Acronyms

Symbols

µC micro-cycle. iii, 6, 7

A

AMQP Advanced Message Queuing Protocol. 5, 7, 16, 19

APN Access Point Name. 17

C

CP Computational Platform. iii, v, 4–7, 15–17, 19, 20, 24, 26, 29, 32, 35, 37, 42

CSMA Carrier Sense Multiple Access. 6, 7

D

DM Data Mining. 7

G

GEO Geostationary Orbit. 9

GPS Global Positioning System. 9, 10, 16

GSO Geosynchronous Orbit. 9

H

HEO Highly inclined Elliptical Orbit. 9

I

IoRT Internet of Remote Things. 12

IoT Internet of Things. 1, 3, 12, 13

J

JSON JavaScript Object Notation. 5, 7, 13, 20, 26, 35

L

LEO Low Earth Orbit. 9

M

MAC Medium Access Control. 6, 7

MEO Medium Earth Orbit. 9

ML Machine Learning. 7

P

PDT Portable Data Terminal. iii, 11, 16

Q

QoS Quality of Service. 12

S

SW Synchronization Window. 6

T

TAW Turn-Around Window. 6

TDMA Time Division Multiple Access. 6, 7

U

USIM UMTS Subscriber Identification Module. 16

V

CHAPTER 1

Introduction

1.1 Motivation

We witness the constant evolution of technology daily, from home appliances to the infrastructures responsible for the world's communications. This evolution allows us to expand and improve the systems that enhance our lives to better fulfill their purpose. If in the past a task like closing the blinds had to be done manually, now, with the support of a simple app and equipment, it can be just a click away, even when away from home.

Moreover, by nature, we are driven by the information we collect from our surroundings, which we use to make better decisions in our daily lives. This information-seeking characteristic is even more evident in the Internet of Things (IoT) context, where the proliferation of sensors allows us to access the data we need in a plethora of scenarios. This concept, firstly mentioned in 1999, describes a system populated with sensors that communicate and exchange information between them. Its applications can be seen in a wide number of areas[1], such as *Smart Homes* (e.g. the control of lights and locks at home), *Smart Cities* (e.g. surveillance and traffic management), and *Smart Agriculture* (e.g. irrigation control and herd supervision).

SheepIT[1] is a practical example of a solution inserted in the agriculture sector, which aims at using herds to control weeds in vineyards as a replacement for current non-ecological methods. These vineyards are typically situated in rural areas with poor network coverage in which the system cannot benefit from its full capabilities — the upload of data to the cloud needs to be executed manually, in an asynchronous and costly process. Furthermore, since these areas are remote or have low population density, the installation of ground-based communications is not considered worth investing in by network service providers. However, the COVID-19 pandemic outbreak has accentuated the need for connectivity in these regions for more than its use in smart systems, since other necessities, such as access to education or remote working, were dependent on it. Thus, satellite connections, which can easily cover wide areas and provide reliable bi-directional connectivity, are a logical solution for these scenarios.

[1] `http://www.av.it.pt/sheepIT` (accessed February 10th, 2022)

This dissertation focuses its work on integrating an existing satellite network in the SheepIT project and developing the adaptations necessary for this change. Thus, it explores a way to take advantage of the satellite's capabilities while complying with its restrictions, allowing the system to benefit from all its functionalities independently of the region where it is located.

1.2 Objectives

Like every other system, SheepIT can and should be improved. This work proposes a solution for its application in remote or poorly covered areas by relying on a new type of communication technology, the satellite. This dissertation had the collaboration of EchoStar Mobile[2], which provided access to its satellite network through a portable data terminal.

The following topics summarize the tasks and objectives of this dissertation:

- Exploring and analyzing the EchoStar Mobile's network and devices as well as the SheepIT's architecture to integrate the new communication technology into the system.
- Analyzing and modifying the type of traffic generated by the SheepIT system to meet the restrictions imposed by satellite communications.
- Optimizing the information exchanged using different serialization mechanisms.
- Testing the impact of the modifications made in the previous tasks.

As a consequence of the work developed to achieve these objectives, the improved system was documented in a paper and submitted for the 10$^{\text{th}}$ International Conference on ICT in Agriculture, Food and Environment — HAICTA 2022.

1.3 Outline

Alongside this introductory chapter, this document presents other four:

- **Chapter 2 – State of the Art**, which addresses concepts that are relevant for this work, namely satellite communications and serialization formats, and presents an introduction of this dissertation's baseline, the SheepIT project.
- **Chapter 3 – Architecture and Implementation**, which presents the system's architecture, and describes the stages and decisions made throughout this work, from the integration of the EchoStar Mobile's network to the optimization of messages exchanged.
- **Chapter 4 – Results**, which presents and analyzes the results of the tests performed in this work to evaluate the impact on the performance of the implemented changes.
- **Chapter 5 – Discussion and Conclusion**, which discusses and presents the concluding thoughts of this dissertation's work as well as relevant improvements for future development.

[2] `https://www.echostarmobile.com` (accessed February 12$^{\text{th}}$, 2022)

CHAPTER 2

State of the Art

This dissertation covers some concepts that need to be addressed to understand the work developed. This chapter serves as an overview of those topics and an exposition of relevant related work. Firstly, it describes this dissertation's starting point — the SheepIT project —, portraying its architecture and interactions. Next, it introduces the definition and purpose of satellite communications and indicates the characteristics of the EchoStar Mobile's network, which was the one used in this work. Lastly, it mentions few existing articles that support the choice made for the serialization formats considered in this dissertation.

2.1 SheepIT Project

SheepIT is an IoT-based system developed to help control and monitor flocks, allowing them to weed cultivated areas, such as vineyards, without affecting the cultures. This project counts with different nodes spread over an area, constantly receiving every animal's information and acting accordingly.

2.1.1 Architecture

The development of this solution had to take some factors into consideration, which resulted in the following characteristics of the system:

- The delimitation of the pasture area is easily defined by shepherds and the system is able to maintain the herd inside it.
- The electronic devices carried by the sheep are light and small, to prevent discomfort, and have long-lasting batteries, allowing a higher autonomy.
- The sheep's posture, in particular, the neck and head position, is monitored to detect undesired behaviors such as feeding on branches of the vines.
- These undesired behaviors are signaled with increasing warning sounds and, if necessary, trigger an electrostatic discharge.
- The sheep's posture and position is obtained frequently and processed locally so that the stimulus can be applied whenever necessary. To do so, a microcontroller is included in the device.

- The data collected and stored about each animal is also transferred to a central computer or cloud service. This information is important to detect anomalous situations such as predator attacks or health problems of an individual.
- The communication infrastructure is able to support thousands of sheep in areas with few hectares while using a reduced number of devices to delimit it. The communication between nodes can consume a great amount of energy and, therefore, the radio technologies and protocols chosen are energy-aware.
- The system is flexible to accommodate eventual expansion and the addition of other sensors.

To make these features possible, the team developed an architecture, illustrated in Figure 2.1, with two main modules: the Wireless Sensor Network (WSN) and the Computational Platform (CP). The former is responsible for all the local tasks, such as the communication between devices and the detection of anomalous situations, while the latter, hosted in a remote location, is responsible for the storage and analysis of the data gathered by the WSN. The communication between these two blocks is supported by a gateway, using the Internet. Additionally, the information was made accessible to the farmers through a user interface.

The following subsections describe the modules aforementioned in more detail.

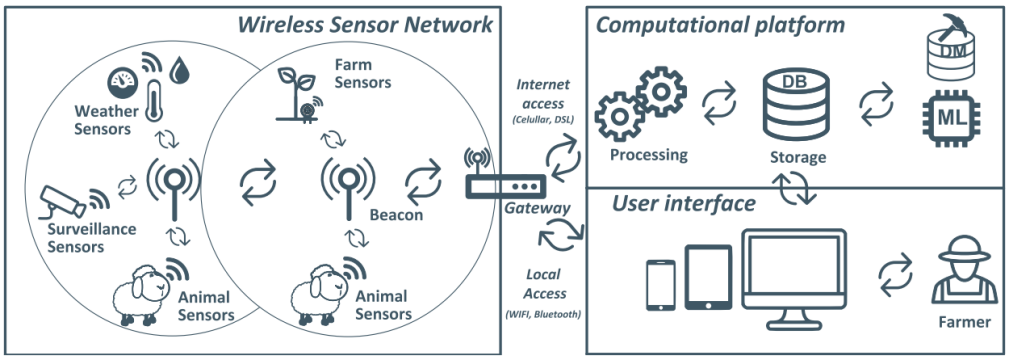


Figure 2.1: Overall system architecture [2].

Wireless Sensor Network

The WSN is a local network and is composed of a set of different types of nodes that communicate between them:

- **Collars** (Figure 2.2 (a)): are carried by the sheep and have a series of sensors, which gather the animal's information and supervise its behavior, and actuators, which act whenever an infraction is detected. Since these nodes are mobile, they are powered by a battery and the radio link included in the collar is used not only to report the data collected but also to estimate the location of the sheep, to minimize the energy consumed.
- **Beacons** (Figure 2.2 (b)): are fixed and are spread strategically throughout the delimited pasture area. These nodes are responsible for collecting the data of the reachable collars and rebroadcasting it to the neighboring beacons, since some of them may be unable to communicate directly with the gateway.

- **Gateway** (Figure 2.2 (c)): usually, there is only one node of this type per pasture, and it can be considered the bridge between the different networks (WSN and CP), allowing them to communicate. It contains a beacon connected to an embedded microcontroller through a serial interface. Due to it being in accessible locations connected to the power grid, there are no restrictions in that regard. Therefore, its hardware is more sophisticated than the nodes aforementioned and, consequently, has higher processing and storage capacities, which enable the integration of the necessary services.

Figure 2.2: Example of the system nodes: Collar (a), Beacon (b) and Gateway (c).

As aforementioned, the gateway is a node with great processing and storage capacity. Thus, the architecture of this type of node is more complex. Figure 2.3 illustrates its architecture, which is composed by the following modules:

- **Serial Port Interface**: allows the communication with the WSN. This interface is responsible for transmission operations, such as the synchronization with the beacon — to send or receive data — and the decoding of the type of message received.
- **Processing Unit**: this module is responsible for the preparation — validation and parsing — of the data received, as well as the localization of the collars with a trilateration algorithm.
- **Shared Memory**: is the memory where the information is stored. It is shared by almost every module of the system.
- **Menu**: it allows the interaction with the collars through the gateway, in which commands can be sent to the collars to change some operations, such as the activation and deactivation of algorithms.
- **Local Web Socket**: enables the communication with a web interface, where the information is displayed.
- **Display**: where the operator can verify the position of the nodes, locally and without accessing the web interface.
- **Alarm Module**: where all the systems' anomalous situations are detected.
- **WSN Information Module**: it maps the data into JavaScript Object Notation (JSON) structures and forwards it to the CP using the Advanced Message Queuing Protocol (AMQP) protocol.

As for its workflow, whenever the gateway receives new information from the WSN, the data is processed and marked with a timestamp. It is then stored in the shared memory and

made available to the other modules. Finally, the relevant fields are sent to the CP, alarm situations are detected, if any, and the information in the web interface is updated.

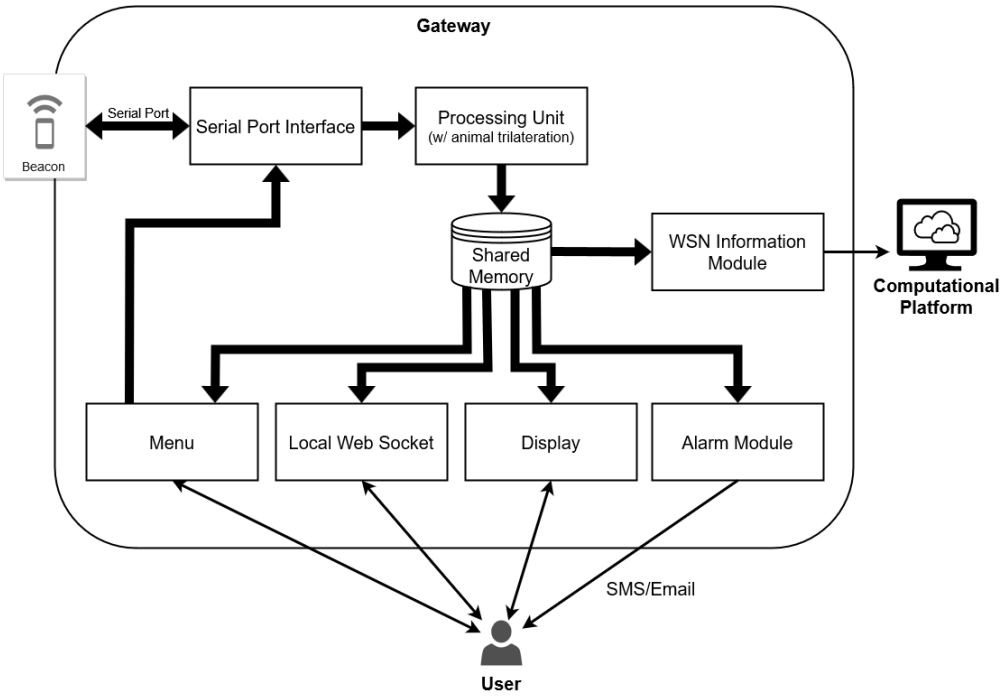


Figure 2.3: Gateway architecture.

The gateway's alarm system is prepared to detect different alarm situations related to animal behaviour and the device's operation. Within the situations related to the animal behaviour, the system can detect infractions — by monitoring the position of the animal's head —, herd disturbances — by comparing the accelerations measured by the collars —, and the absence of an animal — by tracking the time of the last detection of each collar. As for the device's operation, alarms are generated when the battery of a node drops below a minimum threshold, a collar is abandoned, or there is an equipment failure.

As for the communication inside the WSN, a protocol developed by Temprilho, Nobrega, Pedreiras, *et al.*[3] is used. Since the nodes share the same medium, the exchanged traffic is synchronized and organized in periodic time frames (μC). The Medium Access Control (MAC) policies adopted are used according to the message purpose, i.e., for periodic communications it was used Time Division Multiple Access (TDMA), since it was important to avoid collisions and packet losses, while for sporadic communications it was used Carrier Sense Multiple Access (CSMA).

As illustrated in Figure 2.4, each μC is composed of the following components:

- **Synchronization Window (SW)**, where the beacons transmit a synchronization message identifying themselves and informing the type of μC they are sending (Table 2.1);
- **Turn-Around Window (TAW)**, reserved for sensor reading and packet processing;
- **Variable Traffic-type Window (VTW)**, where the different types of traffic are exchanged.

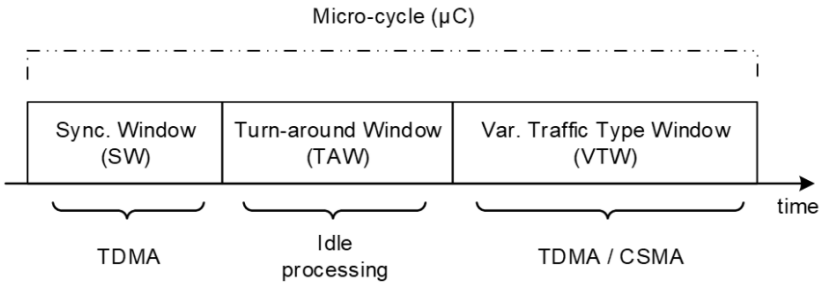


Figure 2.4: μC structure [2].

Table 2.1: Communication micro-cycle types [4].

μC type	Purpose	MAC policy
1	Collar pairing	CSMA
2	Collar communication	TDMA
3	Inter-beacon relay	TDMA

Computational Platform

The CP allows not only a centralization of the data collected by the WSN but also its analysis in order to identify, for instance, behavioral patterns and health issues. To do so, its architecture, illustrated in Figure 2.5, is composed of the following modules:

- **AMQP Broker (RabbitMQ)**: the broker that stores in a queue the messages produced by the WSN in order to be consumed by its subscribers.
- **Processing Framework (Apache Spark)**: the only subscriber of the RabbitMQ queue. It is the main processing framework and is responsible for translating the JSON data structures, processing them with the aid of Data Mining (DM) and Machine Learning (ML) techniques, and storing them in the database. It also includes a rule management module.
- **Data storage (PostgreSQL)**: the database where the information gathered is stored.
- **RESTful API**: used by the user to interact with the system.

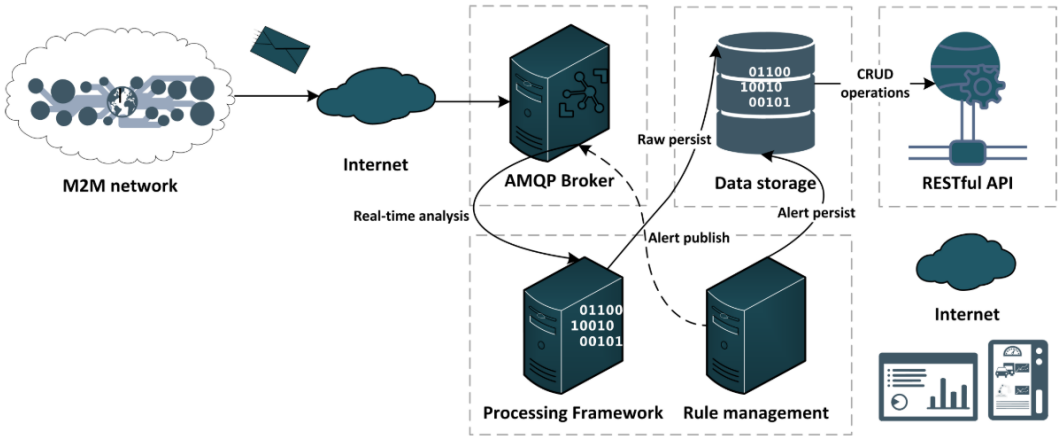


Figure 2.5: Computational Platform architecture [2].

2.2 Satellite Communications

According to Ippolito [5], a communications satellite can be defined as *"an orbiting artificial earth satellite that receives a communications signal from a transmitting ground station, amplifies and possibly processes it, then transmits it back to the earth for reception by one or more receiving ground stations"*. The use of these satellites to provide connectivity between different points on Earth is called satellite communication. Currently, there are several applications for this type of communication, such as broadcasting services — e.g., radio and television —, weather forecasting and navigation.

The classification of the segments of a satellite system depends on the perspective from which it is used. Ippolito [5] divides a satellite communication in a space segment and a ground segment, where the former consists of one or more communications satellites as well as the ground station responsible for their operational control, as illustrated in Figure 2.6, and the latter comprises the earth terminals — fixed, transportable or mobile — that use the space segment's communications capabilities.

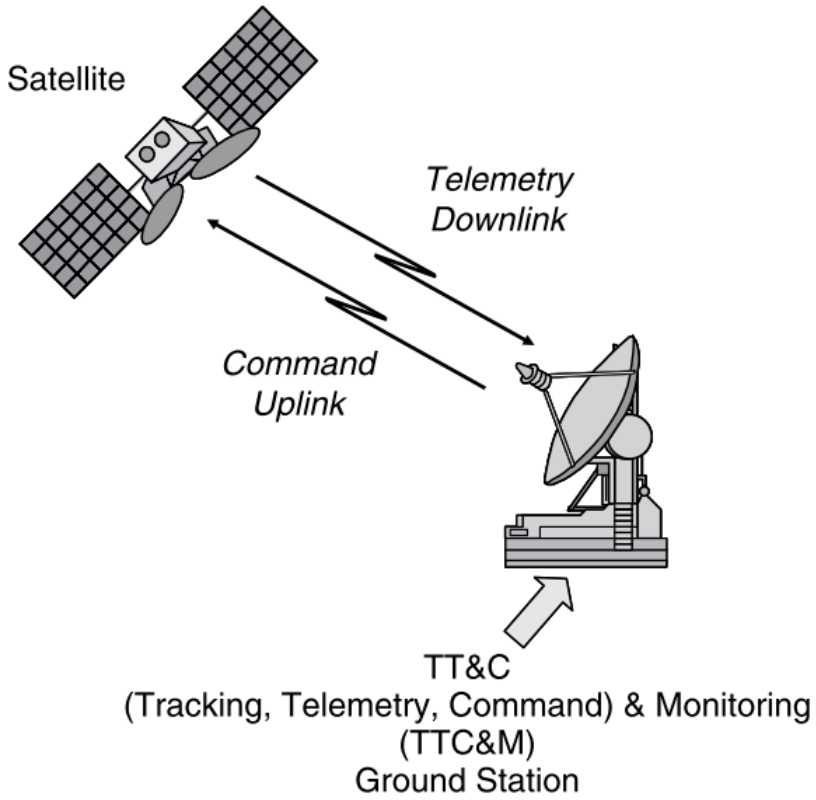


Figure 2.6: The space segment for a communications satellite network [5].

However, at the state-of-the-art level, the possibility of three segments co-existing is also recognized [6]. Figure 2.7 shows this perspective, with two terrestrial segments — the User segment, composed of the user terminals, and the Ground segment, composed of the operator's ground stations —, and a spacial segment — referred to as the Space segment and composed by the satellite constellation.

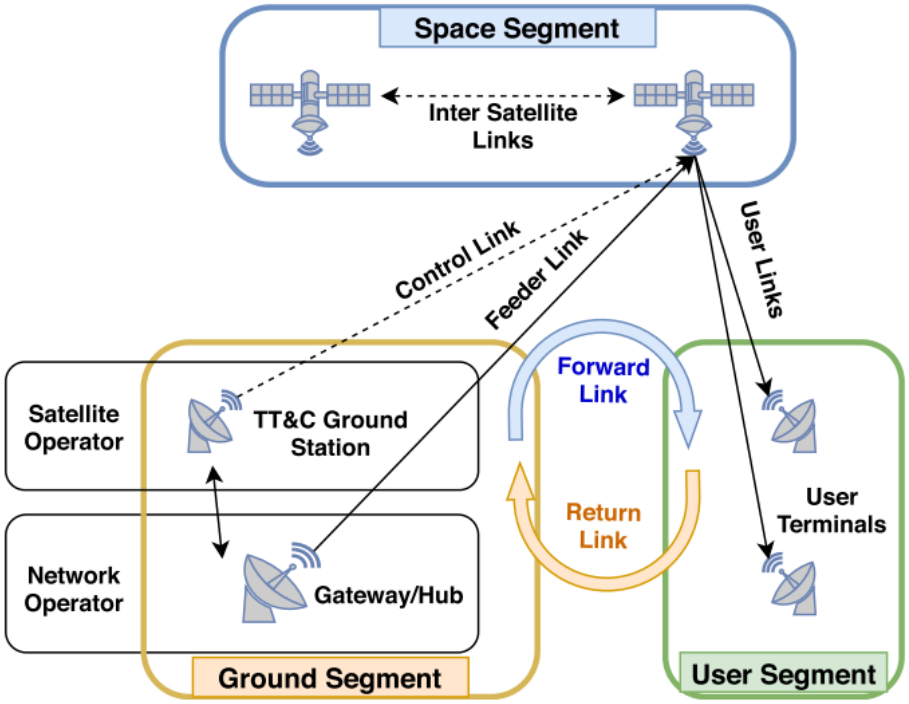


Figure 2.7: SatCom System Architecture [6].

As aforementioned, these artificial satellites orbit our planet. The distance at which they are from Earth determines the nature of their orbit as well as the type of service and coverage they provide. Figure 2.8 displays the four most commonly used orbits, which are described below:

- **Geosynchronous Orbit (GSO)**: situated at, approximately, 36000 km from the Earth's surface, the satellites' speed in this orbit matches the Earth's rotation, having always the same longitudinal coordinates. A Geostationary Orbit (GEO) is a special type of GSO, where the satellite is orbiting the Earth's equator and, consequently, it appears consistently in the same spot in the sky. Their altitude provides them great ground coverage, with only 3 to 4 satellites required for global coverage, and are well-suited for broadcast services. However, their propagation delay is relatively high, rounding the 260 ms.
- **Medium Earth Orbit (MEO)**: is commonly used for navigation systems, such as Global Positioning System (GPS), and is located at an altitude of 5000 to 15000 km. Contrary to GSO, these satellites move across the sky and need to be actively tracked to maintain communications. The propagation delay of these satellites lies between 100 and 130 ms.
- **Low Earth Orbit (LEO)**: located between 500 and 1000 km from the Earth's surface, these are the satellites with the lowest orbit and, consequently, the lowest propagation delay — from 5 to 20 ms. Similarly to MEO, these satellites do not have a fixed position in the sky.
- **Highly inclined Elliptical Orbit (HEO)**: are the only ones with a non-circular orbit,

i.e., their distance to the Earth's surface varies over time. These satellites are suitable to provide coverage to high latitude locations.

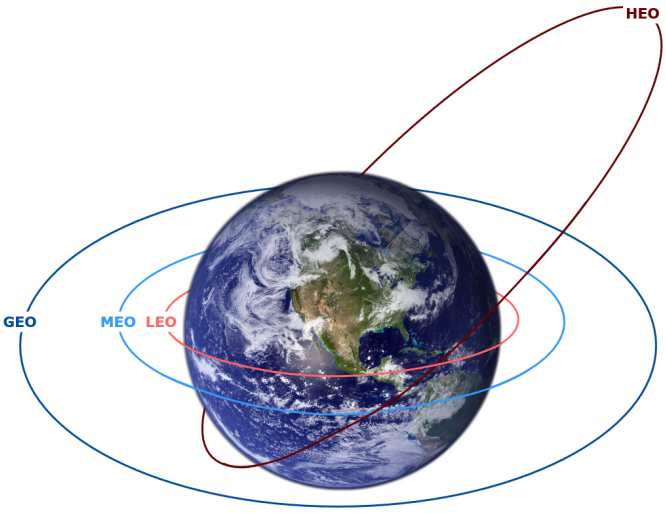


Figure 2.8: Types of orbits.

Satellites communicate with Earth equipment using "*electromagnetic waves to carry information between ground and space*"[7]. The performance of their transmission link depends on the frequency of these electromagnetic waves, which can be more or less affected by the obstacles in their propagation path. Moreover, "*the satellite systems designer must operate within the constraints of international and domestic regulations related to choice of operating free space path frequency*"[5]. Thus, satellites operate in eight different bands from 1 to 50 GHz, illustrated in Figure 2.9. Furthermore, these bands are separated in halves, one for the communication from Earth to the satellite, named uplink, and the other for the communication from the satellite to Earth, the downlink. According to Kodheli, Lagunas, Maturo, *et al.* [6], the L-band is used in radio navigation systems, such as GPS, the S-band in weather and surface ship radars, while TV broadcasting operates predominately in C and K_u bands.

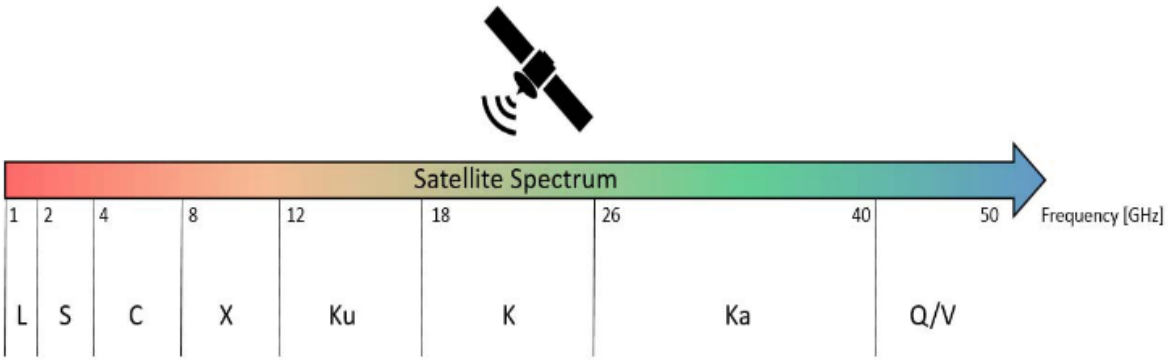


Figure 2.9: Satellite spectrum [6].

2.2.1 EchoStar Mobile

The satellite network used in this work was provided by EchoStar Mobile.

Headquartered in the United Kingdom and subsidiary of EchoStar Corporation — a global provider of satellite communication solutions with a fleet of ten geosynchronous satellites (Figure 2.10) —, EchoStar Mobile is a mobile operator that provides mobile voice and data services across Europe, using the EchoStar 21 S-band satellite. Since these services operate in the S-band, the atmospheric conditions do not have any impact on their performance, which remain stable and available even in adverse weather situations.

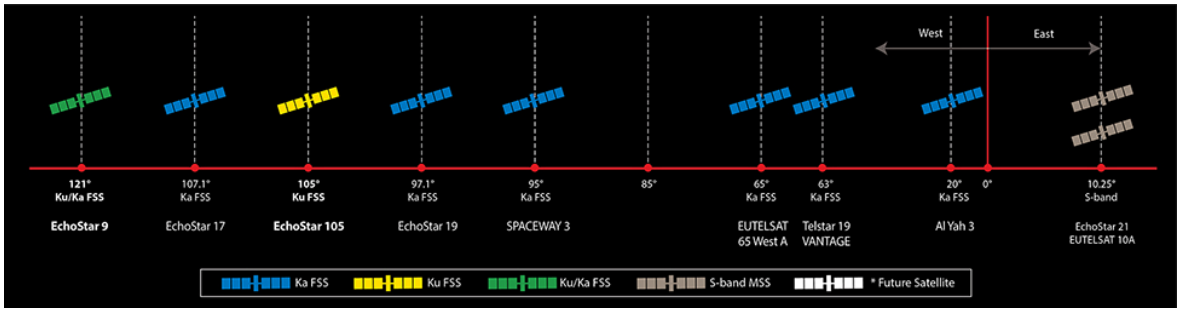


Figure 2.10: EchoStar Satellite Fleet [8].

EchoStar Mobile's services are available for different sectors, such as maritime and agricultural, through one of three Hughes devices:

- Hughes 4510 S-band Terminal
- Hughes 4500 S-band Terminal
- Hughes 4200 Portable Data Terminal (PDT)

For this work, the latter was used. This mobile device, presented in Figure 2.11, provides Internet connectivity through WiFi or Ethernet — with a physical-layer data rate "*up to 290 kbps forward and 256 kbps return*"[1] – and supports voice calls for VoIP users. Moreover, the data transmission cost of this network lies between 4.79 and 9.58 € per Mbit.

Figure 2.11: Hughes 4200 PDT [9]

[1] https://www.echostarmobile.com/satellite-terminals/hughes-4200-portable-data-terminal/ (accessed February 27th, 2022)

2.2.2 Satellite-based IoT

The market research firm IoT Analytics states that the number of connected IoT devices is actively growing, revealing 11.3 thousand million of these devices spread around the world in 2020 and forecasting more than 27 thousand million by 2025[2]. However, some of these devices are located in remote areas that are not covered by ground-based networks. Thus, the role of satellites in the IoT has become more and more pivotal, especially in these situations, since they "*can solve the problem of ground coverage and solve the problem of communication interruption caused by natural disasters*"[10].

The integration of these two technologies is very important and research on the topic has already been done. Article [11] presents three Internet of Remote Things (IoRT) scenarios in which the application of satellites is a key element, and discusses some challenges arising from this integration, such as Quality of Service (QoS) management and heterogeneous networks interoperability. In [12], the authors describe the connectivity limitations of smart farming in remote areas, highlighting their importance in this scenario, and explore satellite communication systems that can provide remote connectivity in Australian smart farms. According to Routray, Tengshe, Javali, *et al.*[13] the hybridization of IoT and satellite networks can also be very important in the context of mission-critical applications. In [14], Qu, Zhang, Cao, *et al.* present a LEO satellite constellation-based IoT system, describing its architecture, routing protocols and heterogeneous networks compatibility, for instance. These papers prove that the use of satellite communications is essential in these scenarios, offering connectivity to IoT-based systems placed in remote areas.

Solutions based on the satellite IoT are currently a reality and its market is expanding globally with applications in different sectors. Optiweigh's product[3] is one of these solutions: a cattle weighing unit with a Swarm embedded modem that tracks the animals' weight and transmits the information to the cloud through Swarm's satellites[4]. Also in the agricultural sector, another animal monitoring solution is available, the Ceres Tag[5]. Ceres Tag uses a smart ear tag with satellite capability — provided by Globalstar's satellite network[6] — to collect animal data, which is transferred to the cloud to be further analyzed. Finally, in the maritime sector, a mobile satellite company named Inmarsat offers a service to ship operators, which allows them to access the data gathered from onboard sensors. This data is also uploaded to a secure database via the Inmarsat network[7].

[2] `https://iot-analytics.com/number-connected-iot-devices/` (accessed February 28th, 2022)

[3] `https://www.optiweigh.com.au/` (accessed March 2nd, 2022)

[4] `https://swarm.space/finding-the-right-iot-connectivity-provider-an-agtech-companys-journey/` (accessed March 2nd, 2022)

[5] `https://www.cerestag.com/` (accessed March 2nd, 2022)

[6] `https://www.globalstar.com/en-gb/blog/case-studies/satellites-and-iot-combine-in-best-of-breed-tr-(1)` (accessed March 2nd, 2022)

[7] `https://www.inmarsat.com/en/solutions-services/maritime/services/fleet-data.html` (accessed March 2nd, 2022)

2.3 Serialization Formats

When compared to other wireless networks, the data transmission cost of a satellite network is higher, therefore it is important to resort to mechanisms that reduce the size of the messages exchanged, such as data serialization.

The concept of data serialization comprehends the transformation of data objects that are part of a complex structure into a sequence of characters (string) or a stream of bytes in order to be stored or sent over the network. The two approaches that are commonly mentioned in the literature are textual if the transformation results in a string, and binary serialization if it results in a stream of bytes. Within each approach, several formats can be chosen, depending on the application's requirements, such as memory and bandwidth constraints or the maximum acceptable latency.

In the context of the IoT, more specifically in smart grid communication, Petersen, Bindner, You, *et al.* [15] compared several serialization formats, both textual and binary, driven by the idea that, depending on the circumstances, there were more adequate formats than the one used by the general communication standards: Extensible Markup Language (XML). To do so, messages similar to the ones transmitted in smart grid use cases were serialized and, in order to also understand the impact of using compression techniques, the serialized messages were compressed. The performance of both the serialization and compression was evaluated based on the time needed for the operations, its memory use and message size. From various conclusions drawn, it was pointed out that, for systems with low-bandwidth, where the most important factor was to reduce the size of the messages, the best serialization formats were MsgPack and Avro, despite being slower. Moreover, although the compression of the serialized messages allowed a slight size reduction, in these cases it was not advantageous as the processing time increased significantly.

A narrower choice of serialization formats was made in the study of Proos and Carlsson [16], where the trade-offs of two Google-developed formats were analyzed in the context of vehicle-to-cloud communication over WiFi or cellular networks. In order to evaluate its performance, real messages from vehicles were used and the metrics chosen for the comparison were the time and memory consumption for serialization and deserialization and the size of the serialized data. For every message, the measurements were performed 1000 times and the average was considered. Proos and Carlsson [16] concluded that the best serialization format for the scenario in question was Protobuf, since it presented a greater serialization speed as well as smaller messages and, consequently, lower latency and bandwidth usage. On the other hand, the deserialization time of Flatbuffers made it a good candidate for scenarios where the communication was reversed (cloud-to-vehicle communications), where the data received in the vehicle needed to be accessed quickly.

An article carried out by Sumaray and Makki [17] compared two textual (XML and JSON) and two binary (Thrift and ProtoBuf) serialization formats in order to optimize the efficiency in a mobile platform. The serialization was applied to two types of messages similar in structure to what could be seen in mobile applications, and the performance was measured

in terms of data size, serialization speed and ease of use. It was concluded that, for storage purposes and when creating new web services from scratch, the binary formats were the best as they provided a greater size reduction and serialization speed. Although the differences between them were minor, ProtoBuf revealed to be faster when deserializing the messages, while Thrift performed better in the serialization. Furthermore, Protobuf presented the smallest messages.

In an embedded environment with great resource constraints, Hamerski, Domingues, Moraes, *et al.* [18] assessed the performance of six implementations of serialization formats: FlatBuffers, ProtoBuf, YAS and three implementations of MessagePack. A producer-consumer application was used to test those libraries using different messages with data types of increasing complexity. The comparison was performed using the serialization and deserialization execution times as well as the data and code sizes. The authors concluded that the best library evaluated was MsgPuck, one of the MessagePack implementations, as it was superior in every aspect considered. Regarding the data size, although the results were slightly worse, the other MessagePack libraries also had a satisfactory performance.

CHAPTER 3

Architecture and Implementation

This chapter describes the work underlying this dissertation's goal of implementing satellite communication capabilities into an existing project and achieving the objectives planned in Section 1.2. The system's architecture is described, as well as the decisions that led to the implementation proposed.

3.1 Architecture

This dissertation's objective consists in the improvement of a smart agricultural system — SheepIT —, by optimizing the messages exchanged and expanding its availability to areas without ground-based network coverage. To do so, the work developed was focused on the connection between the gateway and the CP, where network access is needed.

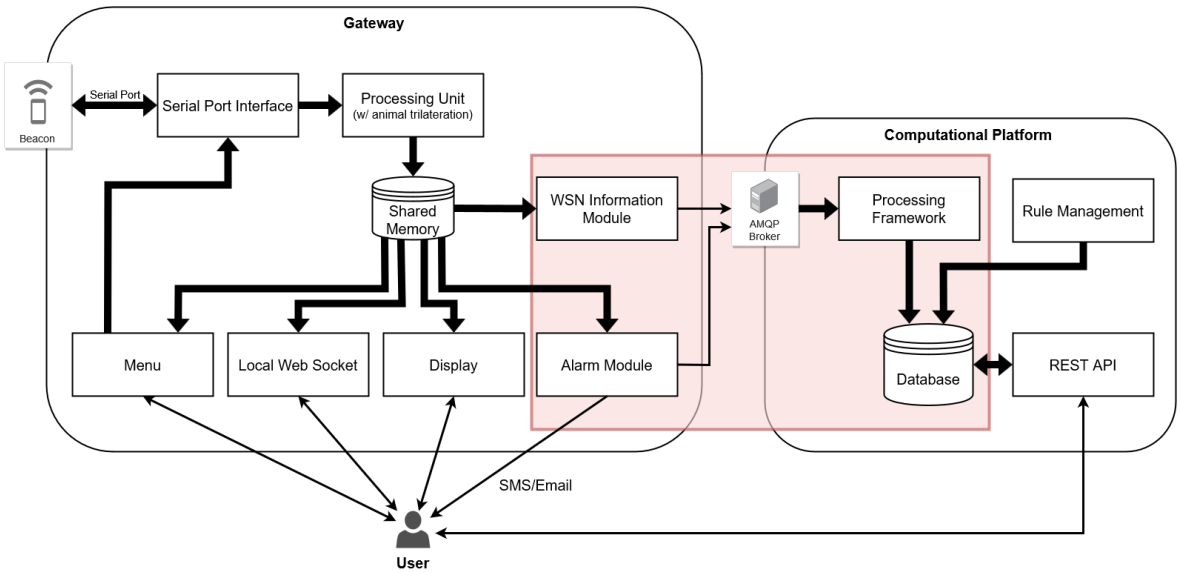


Figure 3.1: System architecture. The red box highlights the modules changed in this work.

Figure 3.1 displays the architecture of the system. From a macroscopic view, the final architecture remains almost the same as the one already described in Section 2.1, with the

exception of a new connection between the "Alarm Module" and the broker. The red box highlights all the modules that required modifications throughout this work.

As previously described, the information gathered in the WSN is transferred to the gateway through a beacon connected to it via a serial port. Then, the data follows a series of processing stages until it is stored and made available for the other modules. This work changed the behavior of two of those modules. The "Alarm Module", which was only responsible for the detection of alarm events, now also uploads that information to the CP. In the case of the "WSN Information Module", the transfer of information to the cloud now only occurs in situations where the communication is not done through a satellite network, which means that this module is only active in non-remote areas. On the CP side, the AMQP broker stores the two types of messages in separated queues, and both are consumed by the "Processing Framework" and stored in the cloud "Database".

3.2 Implementation

3.2.1 Integration of the satellite link

The system uses the satellite communications as an access technology. Thus, as mentioned in Subsection 2.2.1, EchoStar Mobile provided a terminal — Hughes 4200 PDT — to be used in this work to communicate with their network. Although the device also includes a built-in WiFi access point, the interaction with the gateway was achieved using its Ethernet port to minimize the latency between these components. Figure 3.2 illustrates this integration: a gateway is directly connected to the terminal, forwarding messages to the CP through the satellite network.

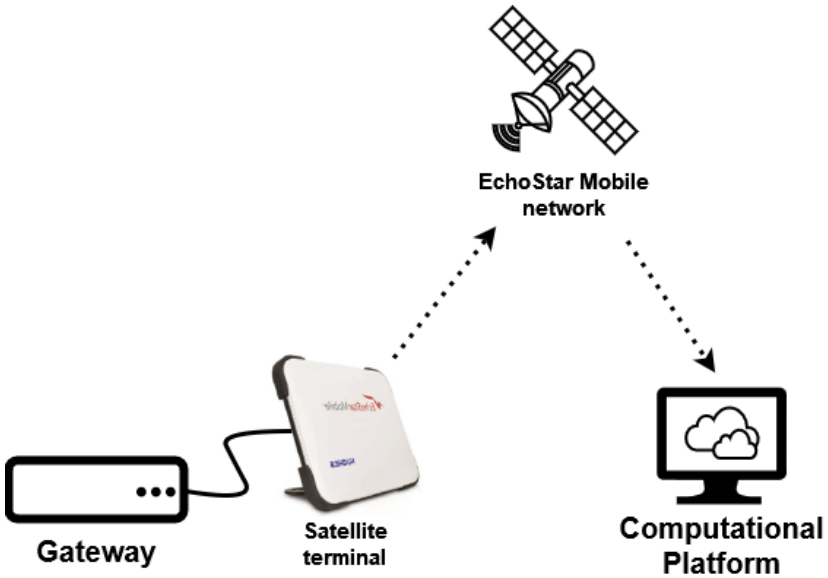


Figure 3.2: Satellite network integration.

The operation of this terminal requires an initial setup. Assuming that the device is powered up and its UMTS Subscriber Identification Module (USIM) and battery are already placed, the first step is to obtain a GPS fix to locate its position. Then, in order to be

connected to the network, an Access Point Name (APN) Profile needs to be defined. Lastly, it is necessary to register the terminal in the network.

As aforementioned, the data transmission cost of a satellite network is higher than other wireless networks, and, in this case, the price per Mbit lies between the 4.79 and 9.58 €. Thus, it is important that this connection is used exclusively for the data transferring to the CP, denying the remaining traffic, since unnecessary traffic can easily lead to an increase of the communication expenses. To do so, the rules presented in Figure 3.3 were created. As can be seen on the left, the traffic with CP's destination address is allowed, while on the right, other information generated in the gateway is blocked.

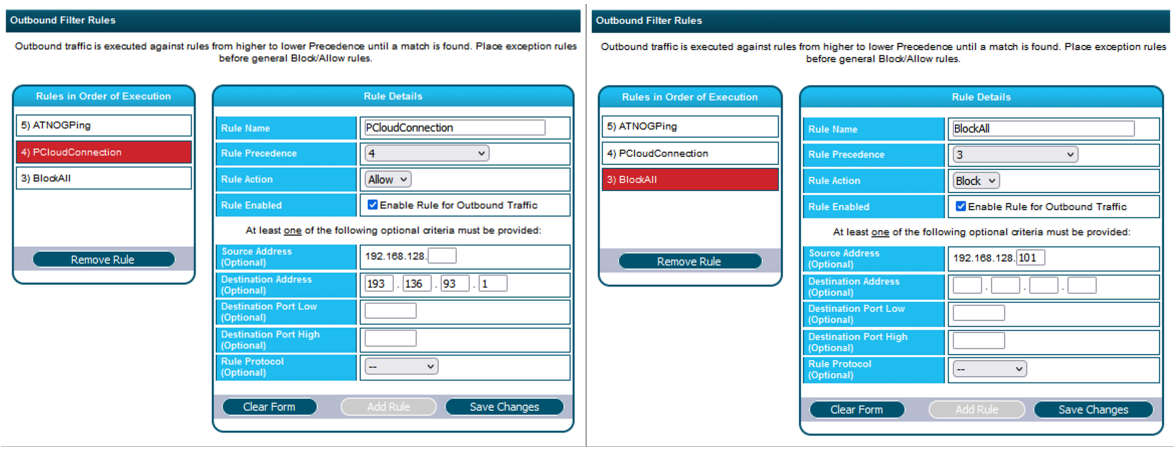


Figure 3.3: Rules created to restrict the traffic of the satellite network.

3.2.2 Alarm manager

After the analysis of the results presented in Section 4.1, it was concluded that the traffic generated by the system needed to be reduced to minimize the communication expenses due to the high transmission cost of the satellite network. To do so, it was decided to alter the information exchanged, i.e., instead of sending the data collected directly by the collars and beacons, only the alarm situations detected by the alarm module are sent. This modification allows not only a lower quantity of data transferred but also enables the users to access the alarms remotely. The following subsections describe the adaptations made both in the gateway and the CP.

Gateway

Since the alarms that are being generated were only stored in a file, the first step was to create a structure that represented an alarm notification in order to be stored in a queue and later be sent to the CP. Considering the examples presented in Table 3.1, taken from a real scenario and before optimization, this structure must contain the following attributes:

- **Timestamp**: the date at which the alarm was generated. Although in the examples this field is represented by strings, its data type was optimized to a *`long`*.

- **Device Type**: the type of device that triggered the alarm. Since there are currently only three types of devices — beacon, collar and herd —, it was used an *unsigned char* to represent it.
- **Id**: the device's identifier. Similar to the examples, it is an *int*.
- **Alarm Type**: the type of the alarm situation. Currently, there is a set of alarm types defined — panic, absence, battery, infraction, lost and equipment malfunction — that, like the device type, can be represented by an *unsigned char*.
- **Priority**: the priority level. There are only three levels: low, medium and high. For this reason, it is only necessary to use an *unsigned char*.
- **Additional Information**: optional field with complementary information. Table 3.1 shows some examples that this attribute may contain. As these illustrate (see first and third rows), notifications with the same alarm type usually contain similar additional information strings. Thus, this element was optimized in order to be represented by an *int*. There are three possibilities: there is no additional information — represented by the number 0 —, there is only one number — represented by itself —, and there are two numbers — which are converted to an int, where the two most significant bytes represent the first number and the two less significant represent the other, as shown in Figure 3.4.

Table 3.1: Examples of alarm notifications.

Timestamp	Device type	ID	Alarm type	Priority	Additional Information
Sat Nov 20 07:22:36 2021	Beacon	1	Battery	Low	Battery: 18 (%)
Sat Nov 20 07:36:12 2021	Collar	2	Infraction	Low	#Warnings = 21 (in 3 min)
Sat Nov 20 08:19:47 2021	Beacon	1	Battery	High	Battery: 6 (%)
Sat Nov 20 08:37:14 2021	Collar	2	Equipment	High	

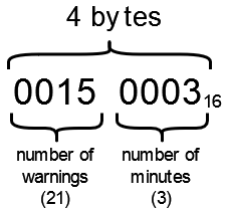


Figure 3.4: Representation of two numbers present in the field "Additional Information", regarding the example from the second row of Table 3.1.

The second step consisted in creating the aforementioned queue and all the necessary methods, such as inserting — required to store the notifications generated —, removing — to delete the alarms already published — and reading — to access the information in the serialization process.

Lastly, when all the algorithms finish their detection, the alarms stored in the queue need to be forwarded to the cloud. Figure 3.5 shows the three stages to do so: initially, the connectivity with the broker where the messages will be published is verified, as well as the existence of notifications in the queue. Afterward, all the alarms are mapped into the

MessagePack[1] format (see implementation in Subsection 3.2.3). Finally, the serialized message is published to the AMQP broker. Note that, in non-remote locations, for instance, with access to WiFi, the system can continue to send the information received by the WSN in addition to the alarms, similarly to the Gateway 2 of Figure 3.6. So, instead of reusing the existing broker queue, a different one was created for the alarms, in order to be possible to differentiate the messages' content.

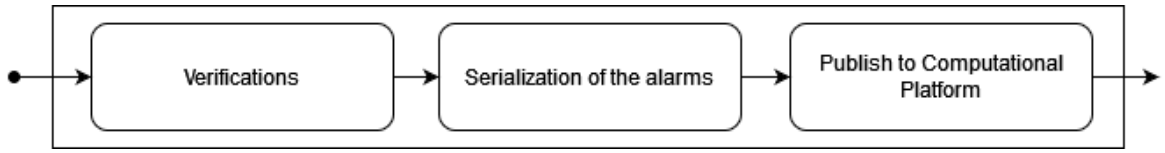


Figure 3.5: The three stages a message follows to be forwarded to the broker.

CP

As aforementioned, another queue was created in the AMQP broker. Therefore, some modifications were made in the CP.

Firstly, the stream responsible for the consumption of the data stored in the broker was transformed into a distributed stream in order to be possible to subscribe both queues.

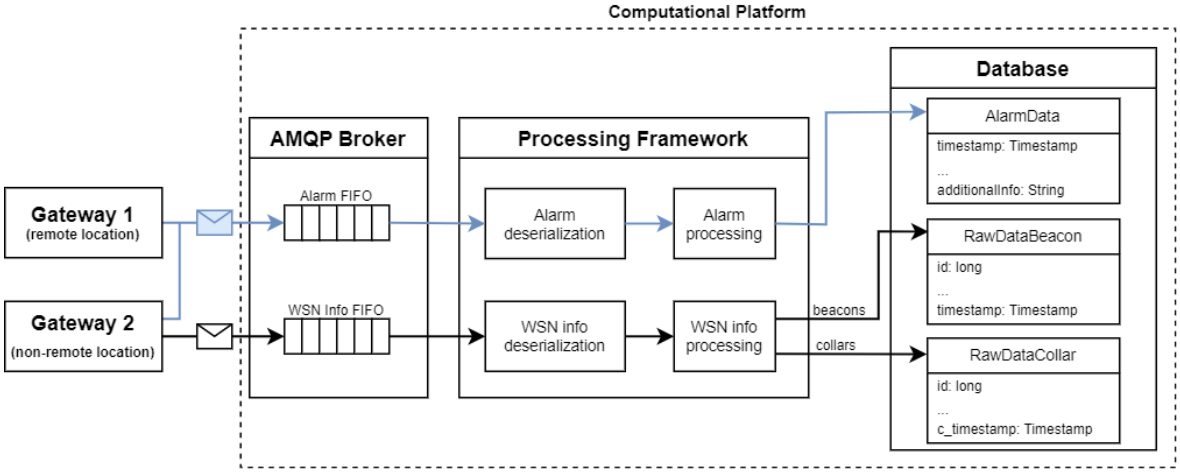


Figure 3.6: Simplified scheme with the CP flow.

Then, as illustrated in Figure 3.6, depending on which queue is read, the message flow can follow the blue arrows, if it was an alarm, or the black arrows, otherwise. For the second case, since most of the modules were already implemented, the only adaptation needed was the creation of the module "WSN info deserialization", where the MessagePack messages were transformed into Beacon and Collar objects. On the other hand, since in the previous version the alarms were not sent, the first case required the creation of all the modules from scratch. For this, it was required to build a new class that represented an alarm with all the methods necessary to handle the particularities mentioned in Section 3.2.2, i.e., the "Additional Information" field decoding and the conversion of the unsigned char attributes into strings.

[1] Apache Avro should have been used instead. Refer to Chapter 5 for more details on this issue.

Then, the "Alarm deserialization" was implemented to convert the MessagePack messages into Alarm objects. Finally, the alarms were stored in the database. To do so, two more elements were created: a class to map the objects into database entries and a table where these entries are stored.

3.2.3 Serialization Formats

This dissertation's baseline used the JSON format to serialize the information sent to the CP. However, this approach is not the best option when using a resource-constrained network, since it is very verbose and, consequently, the size of the messages is unnecessarily large. The use of this format for the encoding of a simple alarm, such as the one in Table 3.2, would result in a message with 100 bytes (excluding the whitespace), as can be seen in Figure 3.7. Therefore, the adoption of a more optimized approach for the mapping of the alarm messages was necessary.

The research presented in Section 2.3 revealed that the best formats for these environments with bandwidth constraints are the MessagePack, Protocol Buffers, and Apache Avro. The following subsections present their characteristics as well as the implementation of each format in this context, describing how they encode the data. To do so, the example of Table 3.2 was considered for comparison.

Table 3.2: Example of an alarm generated by the system. The middle column represents the alarm displayed to the user and the right column represents the corresponding values that will be encoded and sent to the CP.

	Information for the user	**Value to encode**
Timestamp	Wed Apr 6 15:04:02 2022	1649253842
Device Type	Collar	c
Id	5	5
Alarm Type	Panic	p
Priority	Medium	m
Additional Information	Acceleration: 450	450

```
{
        "timestamp": 1649253842,
        "deviceType": "c",
        "id": 5,
        "alarmType": "p",
        "priority": "m",
        "additionalInfo": 450
}
```

Figure 3.7: Example of an alarm represented in the JSON format.

MessagePack

MessagePack is a binary serialization format with support for a plethora of languages, including the ones used in the SheepIT project — C and Java. The libraries applied in this

work are both official implementations[2].

Of the three formats, MessagePack is the only one that does not use any schema to describe how the alarm fields are organized and how they are interpreted. Thus, when performing the serialization and deserialization of messages, the same order must be followed, i.e., the first field that is encoded is the first one to be decoded.

The implementation of MessagePack's serialization in this work consists initially of the creation of a buffer — to store the bytes of the data serialized — and a packer — to convert each field into a sequence of bytes. The first information packed into the buffer is the number of elements that compose the serialized message. Since all the alarms stored in the queue will be sent in a single message, the number of elements is equivalent to the number of alarms times the number of fields in each alarm. Then, for each alarm, the 6 fields — `timestamp`, `deviceType`, `id`, `alarmType`, `priority`, and `additionalInfo` — are serialized according to their data type and added to the buffer.

The implementation of the reverse process — the deserialization — follows the same logic. To do so, an unpacker is created, which is then used to decode the information related to the number of elements the message carries as well as the fields of each alarm. Every set of the 6 aforementioned fields is used to create an alarm object.

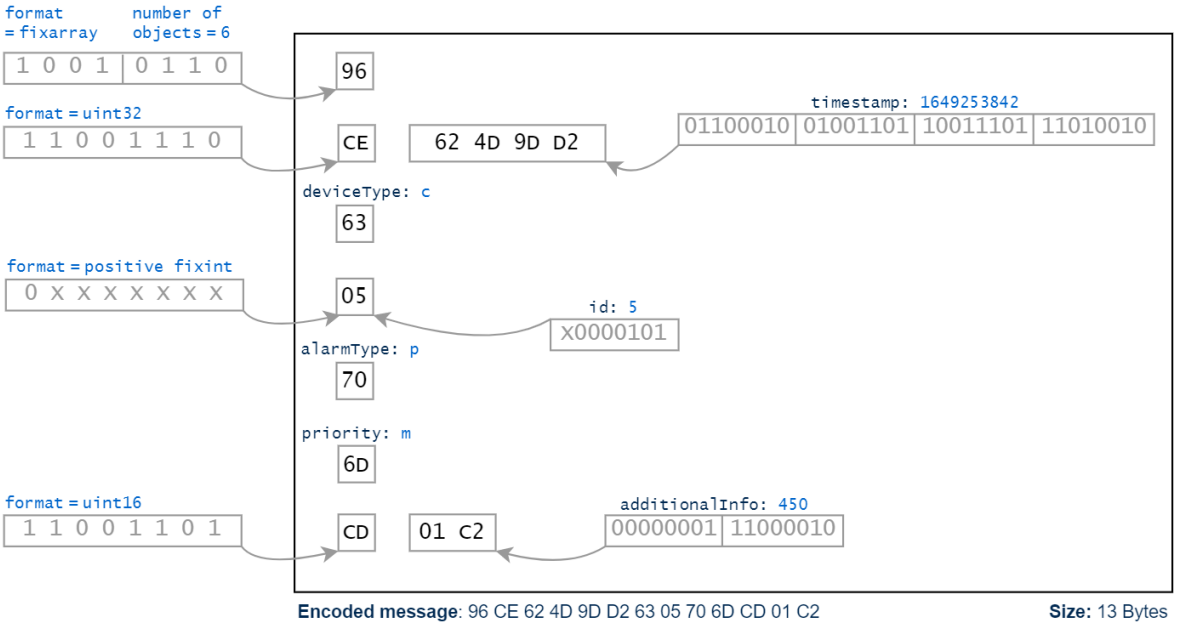


Figure 3.8: Encoding of the alarm message using the MessagePack format.

Considering the example in Table 3.2, since it represents only one alarm, the number of elements is 1 alarm × 6 fields = 6 elements. Figure 3.8 shows MessagePack's strategy to encode this message, which is as follows:

- Preceding the information regarding the elements, MessagePack adds 1, 3 or 5 bytes depending on the length of the array. In this case, the number of elements does not

[2] `https://github.com/msgpack` (accessed May 12th, 2022)

exceed 15 and therefore a single byte is used: the four most significant bits identify the information as a fixarray, and the four least significant bits represent the actual number of elements.

- The encoding of *long* and *int* data types follows a big-endian system. Moreover, MessagePack tags the fields to identify them. These tags depend on the value that is encoded, i.e., even though a field is packed as a *long*, if its value only requires 2 bytes to be represented, then MessagePack chooses the tag corresponding to the format uint16 and encodes the value in just 2 bytes. In this case, the *timestamp* field starts with a byte that labels it as an uint32 and is followed by the encoding of the value. For the *id*, since its binary representation does not exceed 7 bits, the value is encoded in a single byte, where the most significant bit identifies its format — positive fixint. Similarly to the *timestamp*, the *additionalInfo* field uses a byte for its identification — uint16 — and the others for the value.
- For the *char* data type, the UTF-8 character encoding is used. In this case, the *deviceType*, *alarmType* and *priority* only require a byte to encode their information.

The resulting message is the concatenation of the encoded fields. For this example, the serialization of the alarm using MessagePack produces a message with 13 bytes.

After performing the tests of Section 4.2 and concluding that MessagePack was the most suitable format for this scenario, this serialization format was also applied in the exchange of messages related to the WSN information. This process was slightly more complex, since two types of objects need to be sent in the same message — the beacons and the collars. As a way to distinguish which information is being sent, the first packed byte is a *char* with the value of 'a' — for both collars and beacons —, 'b' — only for beacons — or 'c' — only for collars. Then, the flow displayed in Figure 3.9 is followed. Firstly, it is verified if there are any collars stored in the queue. If there are, an array is created — with a size equal to the number of collars times the number of fields in a collar (34) — and each collar is read from the queue and serialized into the array. Then, a similar process is performed for the beacons, with the verification of the existence of beacons, the creation of an array — with a size equal to the number of beacons times the number of fields in a beacon (8) — and the serialization of each one into the array. As for the chosen data types, the majority of the fields were defined as a form of integer — for instance, *int8*, *int16*, *uint8*, *uint16* and *long* —, whose serialization was already described. Moreover, both collars and beacons had two *float* fields. For this data type, MessagePack uses one byte to identify the field as a *float* and four or eight others to encode the 32 or 64-bit values, respectively. The IEEE 754 format is used for this type and the bytes are ordered following a big-endian system.

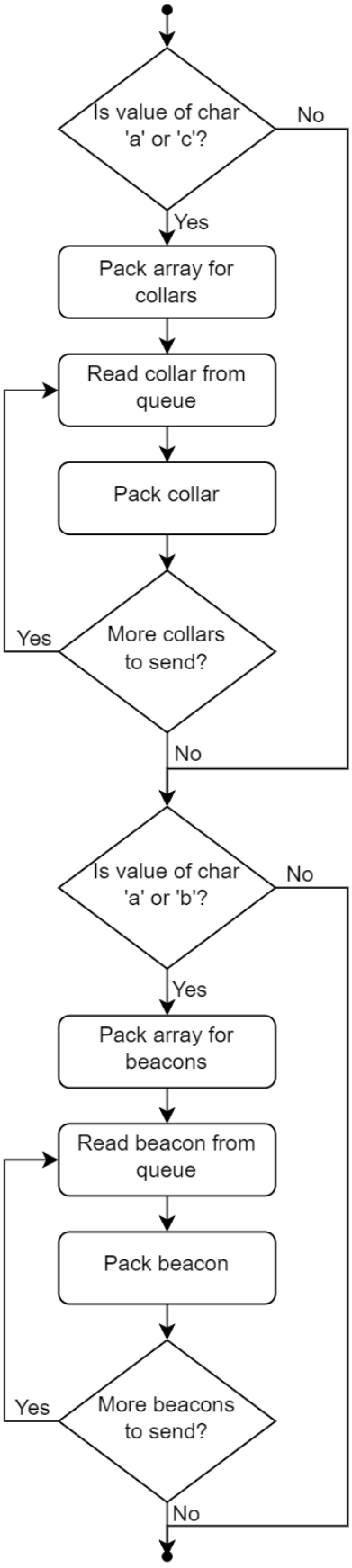


Figure 3.9: Flowchart of the serialization of the WSN information.

Protocol Buffers

Protocol Buffers[3] is a binary serialization format developed by Google and the most commonly used by the company. It supports languages such as Java, Python, and C++. Although the official format does not include C, there is an unofficial implementation made compatible by Dave Benson[4], which was used in this work. Its installation is dependent on the installation of a C and C++ compiler, the official C++ Protocol Buffer runtime and compiler[5], the pkg-config[6], and the autotools (autoconf[7], automake[8], and libtool[9]).

For this format, a schema is required to describe the structure of the message, which is then used for the generation of source code in C — for the gateway — and Java — for the CP. Figure 3.10 displays the schema implemented for this scenario. As mentioned in the previous subsection, several alarms can be sent in the same message. Therefore, two messages were created: the AlarmMessage — which defines the fields of each alarm — and the AlarmMessages — which is composed of 0 or more AlarmMessage messages. The data types available in Protocol Buffers are different from the ones that are used in the C structure. Thus, compatible alternatives were chosen, as can be seen, for instance, in the *timestamp* field, where *int64* is used instead of *long*. Moreover, each field has an associated identifier.

For the implementation of the serialization and deserialization with this format, the functions generated with the schema were used. In both, the process was similar to the one followed in MessagePack. For the serialization, the alarms were read from the queue, one by one, and each field was serialized, producing a single message. For the deserialization, the message was parsed according to the schema and every set of 6 fields was used to rebuild the alarms.

```
message AlarmMessage {
        required int64 timestamp      = 1;
        required bytes deviceType     = 2;
        required int32 id             = 3;
        required bytes alarmType      = 4;
        required bytes priority       = 5;
        required int32 additionalInfo = 6;
}

message AlarmMessages {
        repeated AlarmMessage alarms  = 1;
}
```

Figure 3.10: Protocol Buffers schema used to represent an alarm.

[3] `https://developers.google.com/protocol-buffers` (accessed May 12th, 2022)
[4] `https://github.com/protobuf-c/protobuf-c` (accessed May 12th, 2022)
[5] `https://github.com/protocolbuffers/protobuf` (accessed May 12th, 2022)
[6] `https://www.freedesktop.org/wiki/Software/pkg-config/` (accessed May 12th, 2022)
[7] `https://www.gnu.org/software/autoconf/` (accessed May 12th, 2022)
[8] `https://www.gnu.org/software/automake/` (accessed May 12th, 2022)
[9] `https://www.gnu.org/software/libtool/` (accessed May 12th, 2022)

Figure 3.8 shows the strategy used by Protocol Buffers to encode the alarm in Table 3.2, which is explained as follows:

- Every field starts with a byte that carries two values: its associated number — the numbers 1 to 6 that are provided in the schema — and a wire type. Protocol Buffers groups the different scalar types into 6 wire types. The scalar types of the same wire type are encoded following the same pattern. For this work, two of them were used and are displayed in Table 3.3[10].
- For *length-delimited* fields, two values are encoded in separate bytes: the length, in bytes, of the information in the field plus the actual information. In this case, there are four fields marked as *length-delimited*: *alarms*, *deviceType*, *alarmType*, and *priority*. For all fields, a byte is used to encode the length of the information. The *alarms* field is an embedded message and its length is equal to the sum of the bytes used to encode the fields of the AlarmMessage — 20. As for the other three fields, their length is 1 and their information is the UTF-8 encoding of a character.
- The value stored in the fields indicated as *varint* is encoded in one or more bytes, using a variable-length encoding[11]. In this case, the *timestamp* is serialized into 5 bytes, the *id* into 1, and the *additionalInfo* into 2.

Akin to MessagePack's process, the resulting message is the concatenation of each encoded field. For this format, the encoding of the alarm presented in Table 3.2 results in a 22-byte message.

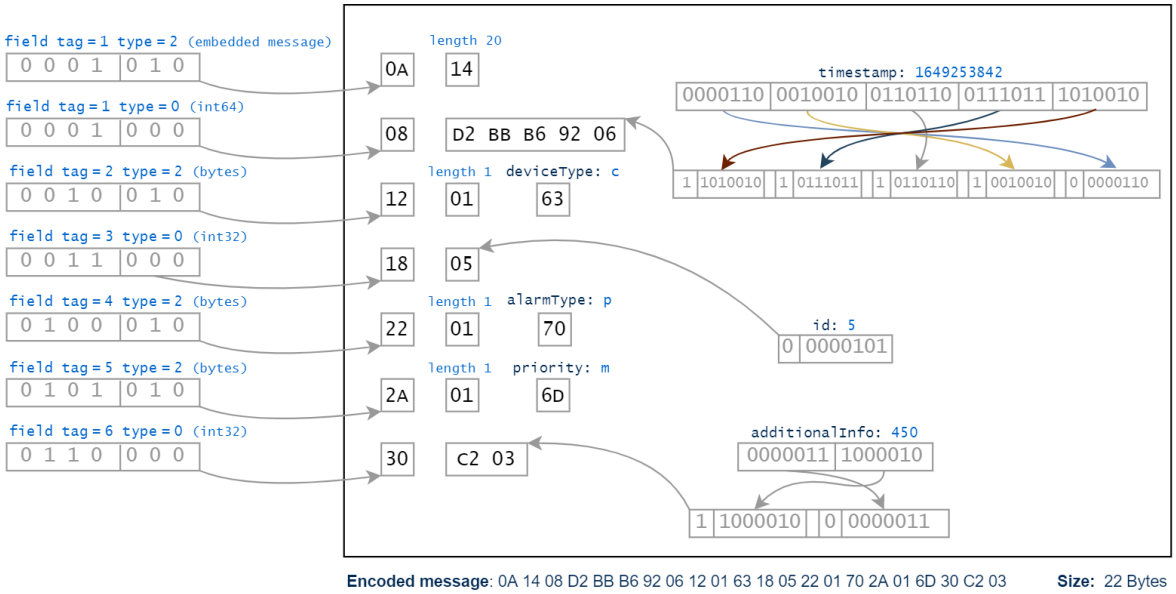


Figure 3.11: Encoding of the alarm message using the Protocol Buffers format.

[10] See the complete table in `https://developers.google.com/protocol-buffers/docs/encoding`

[11] In this type of encoding, the most significant bit of every byte is used to indicate whether or not further bytes exist. Thus, the value is split in groups of seven bits. The most significant byte that is encoded holds the least significant group.

Table 3.3: Wire types used in this work.

Wire Type	Meaning	Used for
0	varint	int32, int64, ...
2	length-delimited	bytes, embedded messages, ...

Apache Avro

Similarly to MessagePack and Protocol Buffers, Apache Avro[12] is a binary serialization format with support for multiple languages, including C and Java. For this work, the official implementations were used. The installation of the C library also required the installation of a dependency — the Jansson JSON parser[13].

This format works with schemas, which are defined in JSON. In contrast to Protocol Buffers, the generation of source code is not a requirement. Nevertheless, the same schema needs to be available at both endpoints so that the information is correctly handled. The schema developed for the representation of the alarms is shown in Figure 3.12. In this format, the *char* data type is not available and therefore the *bytes* type was chosen to represent the *deviceType*, *alarmType* and *priority* fields. The remaining fields were defined accordingly with the data types used for the C structure — *int* and *long*.

```
{
        "type":"record",
        "name":"AlarmMessage",
        "fields": [
                {"name":"timestamp", "type":"long"},
                {"name":"deviceType", "type":"bytes"},
                {"name":"id", "type":"int"},
                {"name":"alarmType", "type":"bytes"},
                {"name":"priority", "type":"bytes"},
                {"name":"additionalInfo", "type":"int"}
        ]
}
```

Figure 3.12: Apache Avro schema used to represent an alarm.

For the implementation of the serialization and deserialization of Apache Avro, the schema is read from an Apache Avro Schema file and parsed into a schema data structure. The remaining process is similar to the other formats: in the gateway, each alarm is read from the queue and its fields are encoded with the help of the schema; in the CP, the fields are decoded and used to restore the original alarm messages.

Figure 3.13 demonstrates how Apache Avro encodes the example of Table 3.2, which is as follows:

- *Long* and *int* fields are handled in the same manner. The encoding of their value requires two steps, since they use a combination of two types of encoding — zig-zag[14]

[12] `https://avro.apache.org/` (accessed May 12th, 2022)
[13] `https://github.com/akheron/jansson` (accessed May 12th, 2022)
[14] This type of encoding allows negative numbers with small absolute value to also have a small encoded

and variable-length[11]. In this case, since the three fields of this category are positive — *timestamp*, *id*, and *additionalInfo* —, the value is shifted 1 bit to the left and split into groups of seven bits, which are then reorganized. The *timestamp* is encoded into 5 bytes, the *id* into 1 and the *additionalInfo* into 2.

- For the fields defined as *bytes*, two values are encoded. The first value holds a *long* with the length, in bytes, of the information encoded. The last value holds a sequence of bytes of information. In this case, the *deviceType*, *alarmType*, and *priority* fields are composed of only a byte of information, which is the UTF-8 encoding of a character. Therefore, these fields are encoded in 2 bytes.

Similarly to the other formats, the produced message is the concatenation of the encoded alarm fields. The encoding of this example using Apache Avro produces a binary message with 14 bytes.

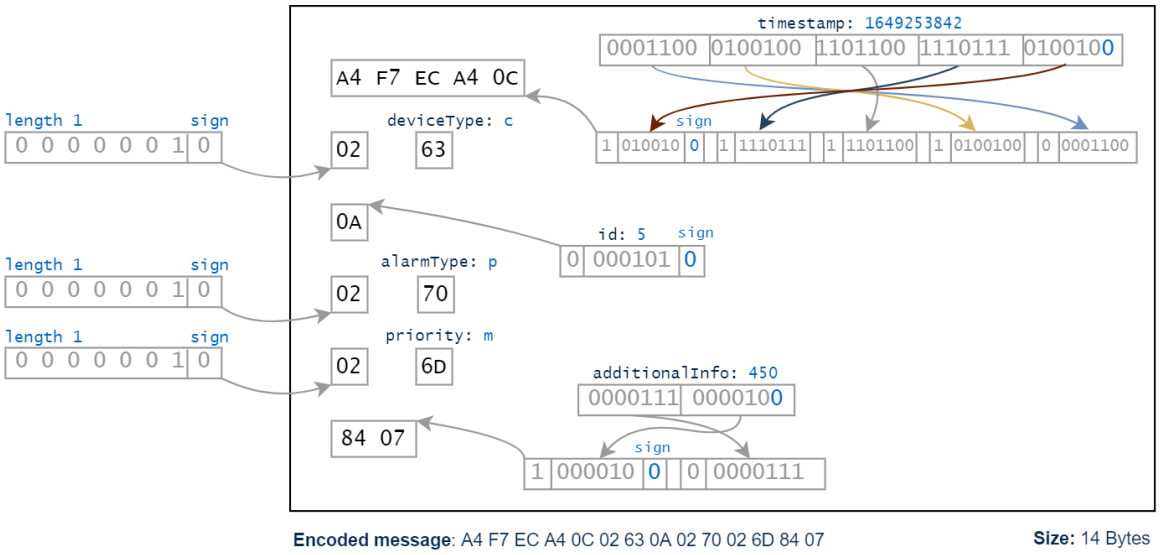


Figure 3.13: Encoding of the alarm message using the Apache Avro format.

value by mapping signed integers into unsigned integers. Considering a number n with 32 bits, the value is encoded using $(n << 1) \wedge (n >> 31)$. For positive numbers, only the left shift is necessary.

CHAPTER 4

Results

Throughout the development of the work described in Chapter 3, several tests were performed, which are presented, along with their results, in the following sections. This chapter analyzes the system before and after the implemented changes in terms of the volume of information exchanged as well as its latency, measured through a network sniffer. The three serialization formats previously mentioned are compared in order to determine the one with the best performance. Lastly, the processing time of the modules modified is analyzed.

The generation of different volumes of data was simulated using a fictitious client of the gateway. For the majority of the tests, only a virtual machine hosted on a computer with 8GB of RAM and a dual-core processor was used to run the gateway. The temporal analysis tests were also performed using a Raspberry Pi 3 Model B+ in addition to the mentioned virtual machine.

4.1 Integration of the satellite link

4.1.1 Setup

Before the integration of the satellite communications interface, the performance of the system was tested. To assert the need of optimizing the information uploaded to the CP, the first test consisted on the measurement of the volume of data produced. Then, the latency of the system with WiFi connectivity was measured. After the satellite connection was established, the latency test was repeated. The messages used in both tests simulate the transport of the collar information received by the WSN.

Regarding the first test, the periodicity at which the system uploads the data can be configured to transfer more or less information daily. Moreover, different systems can monitor a different number of animals. Thus, these two metrics were used to assess the daily volume of traffic exchanged. The tests were run with the following values:

- **Number of collars**: 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000
- **Periodicity (s)**: 30, 60, 90

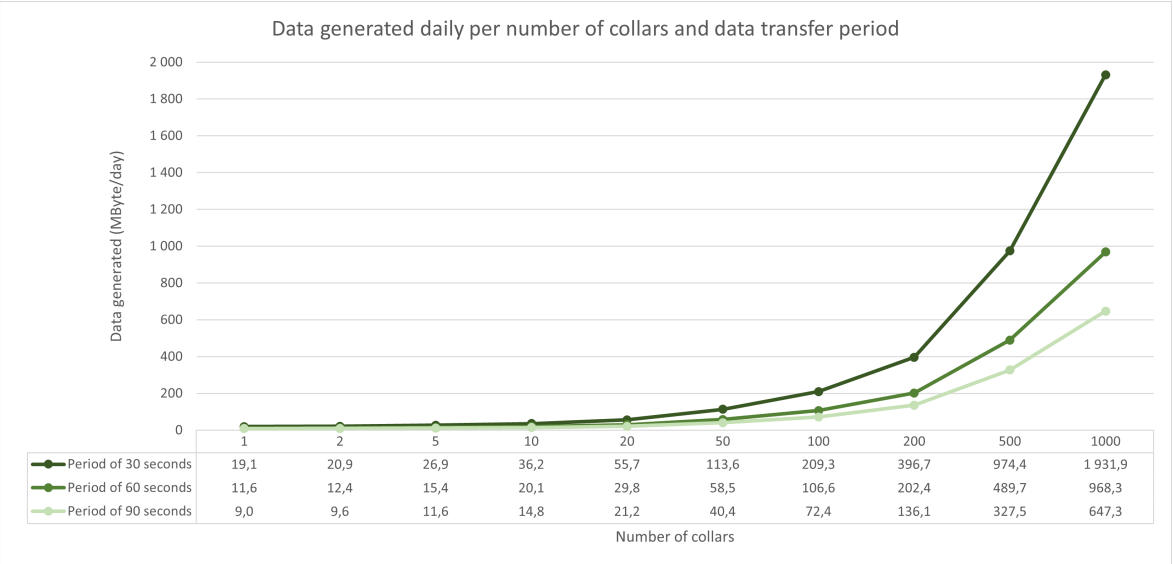
In this scenario, the same amount of messages was simulated for each combination. Thus, the duration of the experiment varied according to the periodicity, with higher duration for the period of 90 seconds. From the results obtained, the amount of data generated in a 24-hour interval was extrapolated and its cost was calculated.

Regarding the second test, the latency of the system for the transmission of the collar information was registered at different times of the day, both using WiFi connectivity and the EchoStar's satellite link. This experiment was repeated 10 times for both 1 and 100 collars. Moreover, the results were registered with a 95% confidence interval, using a t-distribution.

4.1.2 Analysis

As expected, the results of the first test show an increase in the amount of data generated with the increase of the number of collars and the decrease of the period, as can be seen in Figure 4.1. In these conditions, the maximum volume of data is achieved for 1000 collars and a period of 30 seconds, with a daily volume of 1931.9 MB $\approx$ 1.89 GB. Regarding the maximum data rate allowed by this technology — 290 kbps, as stated in Subsection 2.2.1 —, it is possible to send up to $290 \div 8 \times 86400 = 3132000$ kB $\approx$ 2.99 GB daily. Therefore, the data rate provided by this network is sufficient for the volume of data generated by the system.

However, considering the data transmission cost of this network — of 4.79 to 9.58 € per Mbit, as previously mentioned —, even the unrealistic scenario of a herd with a single sheep would have a significant cost: for a period of 90 seconds, the daily cost would be between $9.0 \times 8 \times 4.79 \approx 345$ € and $9.0 \times 8 \times 9.58 \approx 690$ €. These considerably expensive values would be even greater for larger herds as they are directly proportional to the values of data generated displayed in Figure 4.1. As such, the increase of the data transfer period is not a viable solution, since the volume of data produced continues to represent a great expense for the system. Therefore, the optimization of these messages is required.



	1	2	5	10	20	50	100	200	500	1000
Period of 30 seconds	19,1	20,9	26,9	36,2	55,7	113,6	209,3	396,7	974,4	1 931,9
Period of 60 seconds	11,6	12,4	15,4	20,1	29,8	58,5	106,6	202,4	489,7	968,3
Period of 90 seconds	9,0	9,6	11,6	14,8	21,2	40,4	72,4	136,1	327,5	647,3

Figure 4.1: Impact of the number of collars and data transfer period on the amount of data generated.

As for the results of the second test, Figure 4.2 shows the variation in the time taken to send information. As can be seen, in both cases the latency increases with the increase of the message size. Both technologies show some fluctuation throughout the day, although this would have no impact when applied in a realistic scenario.

Moreover, the satellite connection presents a considerably greater latency time than the WiFi, due to two major factors. Firstly, the path taken by the messages is not the same in each technology. As shown in Figure 4.3, the information sent via the satellite link travels to the destination using a greater number of hops. Lastly, the data rate available for the satellite connection is limited. Thus, the same information takes more time to be uploaded in the case of this connection.

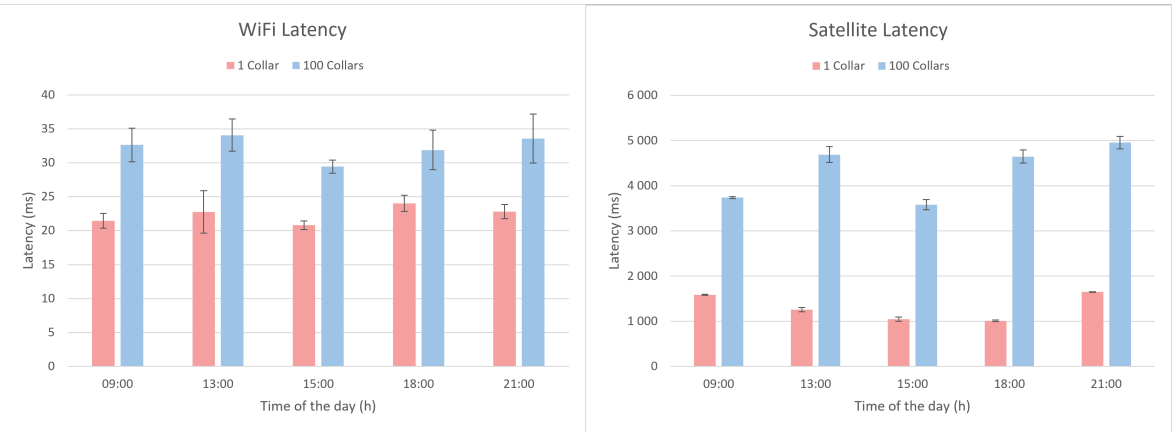


Figure 4.2: Latency of the system using a WiFi network (left) and a satellite network (right).

```
root@sheepit:~# tracepath atnog.av.it.pt                                    a
 1?: [LOCALHOST]                      pmtu 1500
 1:  vodafonegw                                            2.459ms
 1:  vodafonegw                                            1.290ms
 2:  no reply
 3:  107.41.30.213.rev.vodafone.pt                        10.790ms asymm  7
 4:  FCCN.AS1930.gigapix.pt                               10.604ms asymm  7
 5:  Router30.Lisboa.fccn.pt                              14.629ms asymm  9
 6:  Router20.Lisboa.fccn.pt                              15.128ms asymm  8
 7:  Router41.Porto.fccn.pt                               15.574ms
 8:  UA.Aveiro.fccn.pt                                    15.525ms
 9:  193.137.173.234                                      16.859ms
```

```
root@sheepit:~# tracepath atnog.av.it.pt                                    b
 1?: [LOCALHOST]                      pmtu 1500
 1:  10.131.1.71                                           1.420ms
 1:  10.131.1.71                                           1.345ms
 2:  no reply
 3:  10.115.203.9                                        824.237ms
 4:  10.115.203.25                                       924.529ms
 5:  10.110.203.1                                        793.023ms
 6:  10.110.203.17                                       793.343ms
 7:  10.110.64.24                                        793.208ms
 8:  no reply
 9:  10.110.64.24                                        721.123ms asymm  7
10:  10.110.65.1                                         792.782ms
11:  159.100.54.241                                      797.609ms
12:  159.100.54.233                                      936.156ms asymm 13
13:  31.217.132.89                                       935.893ms
14:  ae0-3132.bb.the.lon.as43531.net                     821.032ms asymm 13
15:  ve47.core3.lon1.he.net                              757.112ms asymm 14
16:  100ge0-33.core2.lon3.he.net                        1123.492ms asymm 15
16:  100ge0-33.core2.lon3.he.net                        1799.929ms asymm 15
17:  100ge4-1.core1.lis1.he.net                          895.893ms asymm 16
18:  FCCN.AS1930.gigapix.pt                              961.877ms asymm 17
19:  Router30.Lisboa.fccn.pt                             932.066ms
20:  Router43.Backbone2.Porto.fccn.pt                    931.824ms
21:  ROUTER42.10GE.Porto.fccn.pt                         776.623ms
22:  UA.Aveiro.via-Porto.fccn.pt                         771.691ms
23:  193.137.173.234                                     902.425ms asymm 24
```

Figure 4.3: Path taken to communicate with a server hosted at Instituto of Telecomunicações through a WiFi (a) and satellite (b) connections.

4.2 Selection of the best serialization format

4.2.1 Setup

To evaluate the best solution for the optimization of the messages exchanged between the gateway and the CP, three serialization formats were tested, namely Apache Avro, MessagePack and Protocol Buffers. To do so, for each existing type of alarm, a varied number of messages was generated through an alarm simulator created in the gateway. Table 4.1 shows the different amounts tested. The maximum number of messages simulated for the two types of devices is different, since a real system uses a reduced number of beacons to cover

the pasture area while monitoring a herd that is typically composed of a greater number of animals.

The tests were repeated 10 times for each combination and the average was registered with a 95% confidence interval, using a t-distribution. The performance of the encoding APIs was measured regarding the message size. The latency of the system in the upload of alarms was also tested for the format with the best performance.

Table 4.1: Number of alarm messages tested for each type of device.

Type of device	Number of messages
Collar	1, 2, 10, 50, 100, 500
Beacon	1, 2, 5, 10, 15, 20

4.2.2 Analysis

Although every type of alarm was represented by the same data structure, Figure 4.4 shows that the messages produced by them have different sizes, revealing a relationship between these two metrics. Moreover, for the simulation of 10 alarms, MessagePack produced the lowest volume of data for almost every type of alarm, while Protocol Buffers was clearly the one with the worst results.

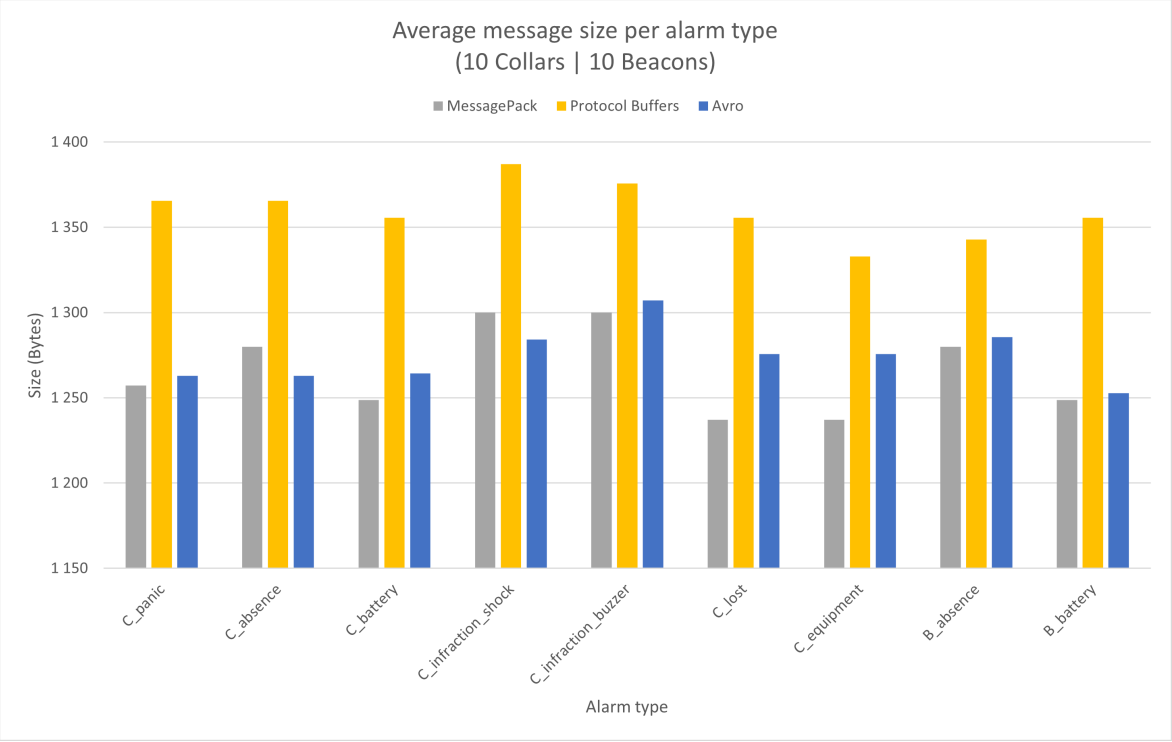


Figure 4.4: Comparison of the average message size of the different alarm types.

The impact of the number of alarms on the size of the messages produced by the serialization formats is slightly different for alarm types with smaller and larger messages, as can be seen in Figure 4.5 and Figure 4.6, respectively. In the former, the growth of the message size starts to be distinct from the 50 alarms upwards for all formats, but with MessagePack presenting

the best size reduction. In the latter, both MessagePack and Apache Avro have a similar behaviour.

Overall, MessagePack was the format that provided the greater size reduction and, therefore, achieved the best performance.

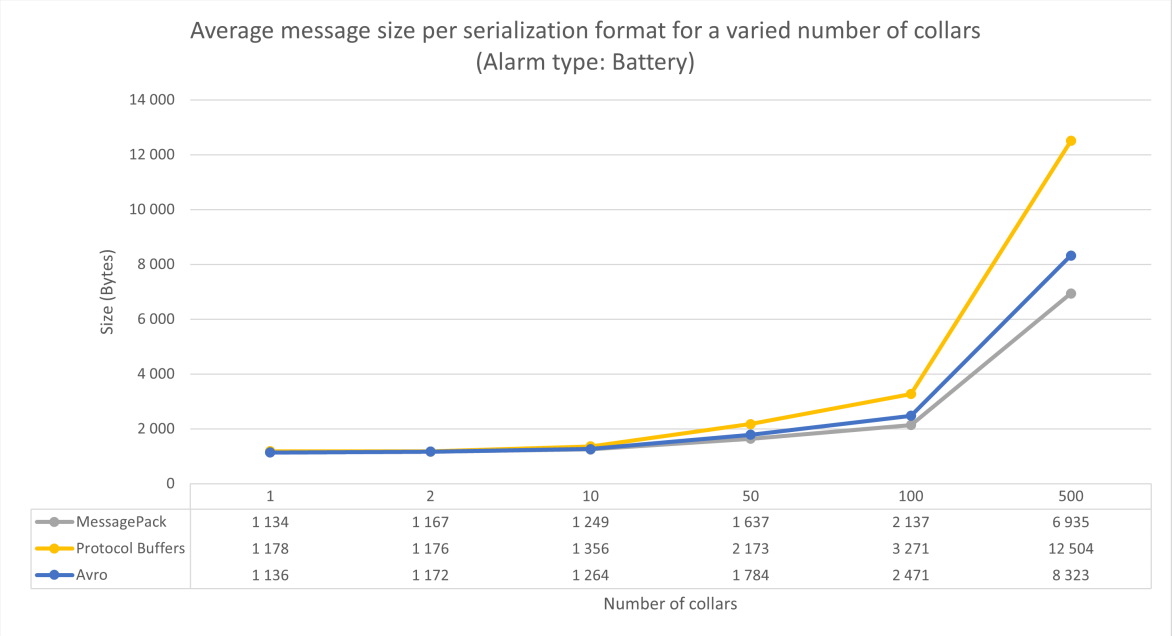


Figure 4.5: Impact of the number of collars on the serialization formats' message sizes - Battery alarm.

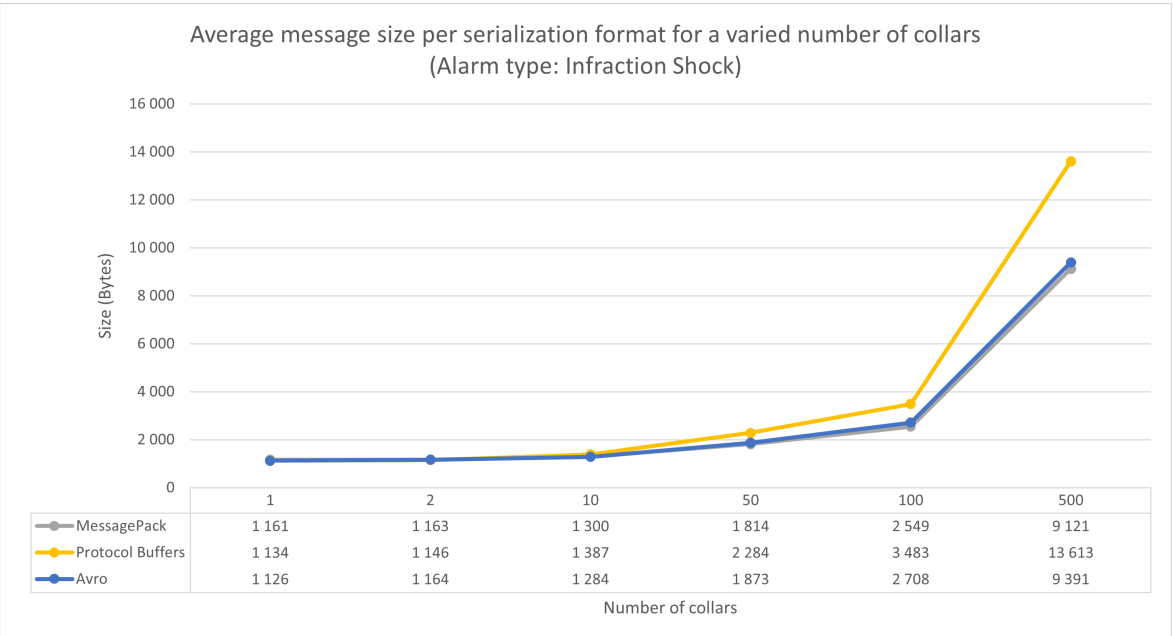


Figure 4.6: Impact of the number of collars on the serialization formats' message sizes - Infraction shock alarm.

Moreover, Figure 4.7 shows the evolution of the system's latency in the transport of two alarm types using the MessagePack format. As expected, the alarm types that produce bigger

messages also take longer to transfer the information. Comparatively to the initial system, the modification and optimization of the information uploaded allows the reduction of the sending time from an average of approximately 4 seconds to less than 1.6 seconds, regarding the information of 100 collars.

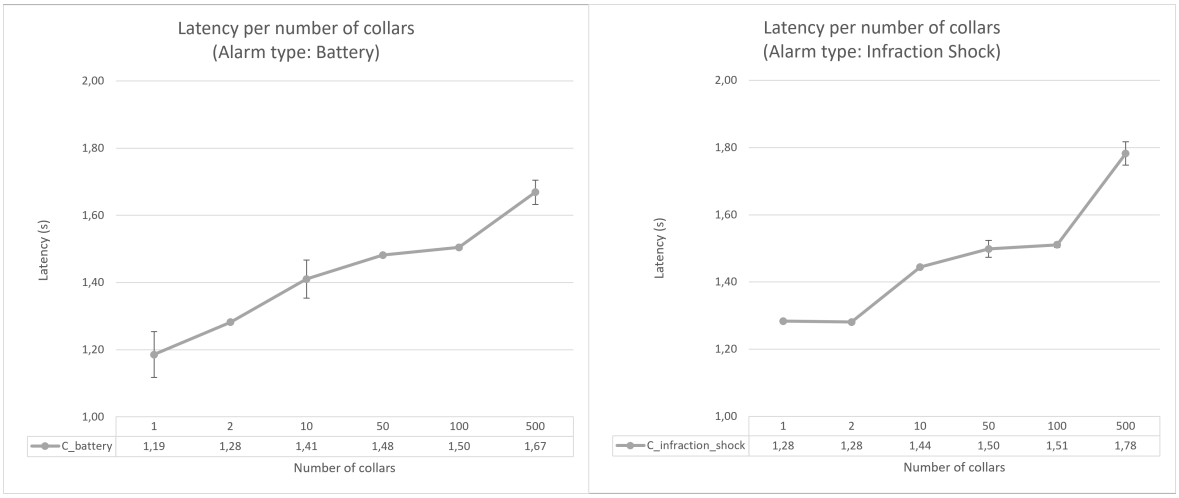


Figure 4.7: Impact of the number of collars on the messages' latency for the battery (left) and infraction shock (right) alarms.

4.3 Optimization of the WSN information

4.3.1 Setup

The test of the previous section revealed that MessagePack was the adequate format to encode the information forwarded to the CP. Thus, this format was also applied to the part of the system that sends the WSN information, which was previously encoded in JSON. To assess the impact of these changes, two metrics were measured: the volume of traffic produced and the corresponding latency.

The conditions of both tests were similar to the ones in Section 4.1, with the exception of the periodicity for sending the information, which was set at 60 seconds. These results were compared to the results obtained before the system's optimization.

4.3.2 Analysis

Figure 4.8 compares the volume of traffic generated in the system before and after the optimization. As can be seen, the modification of the message encoding allowed a significant reduction on the amount of information exchanged daily, with approximately 10 times less data from the 100 collars upwards. Regarding the data transmission cost, the simplest scenario of a herd with a single sheep continues to represent a great daily expense, although with lower values — from $2.9 \times 8 \times 4.79 \approx 111$ € to $2.9 \times 8 \times 9.58 \approx 222$ €.

This optimization has also an impact on the latency of the system. Figure 4.9 shows the evolution of the information sending time, reaching 4 seconds for the maximum number of collars tested. Compared to the results of Figure 4.2 regarding the satellite network, the latency for 100 collars decreased to, approximately, 2.9 times less.

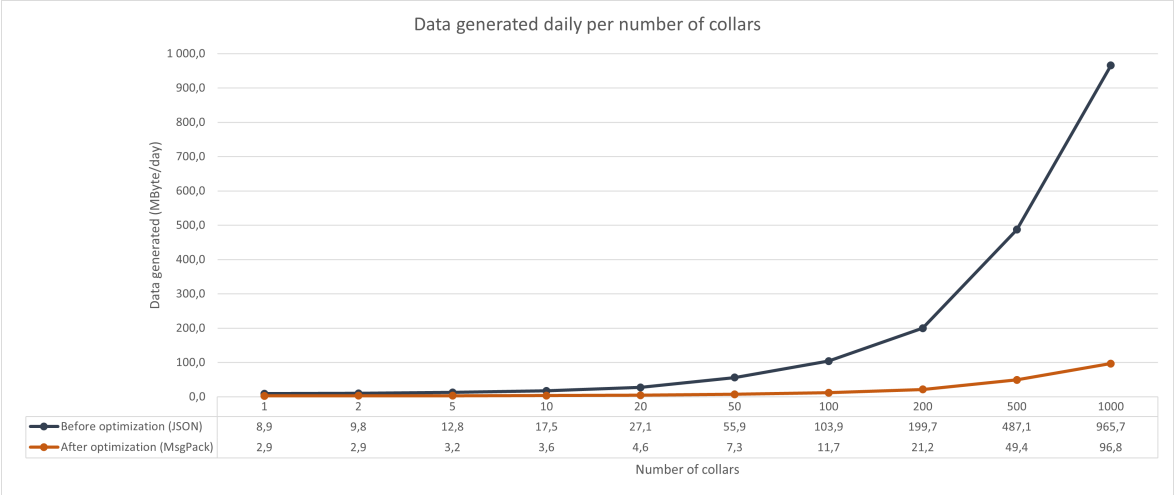


Figure 4.8: Comparison of the data generated before and after optimization with the MessagePack serialization format.

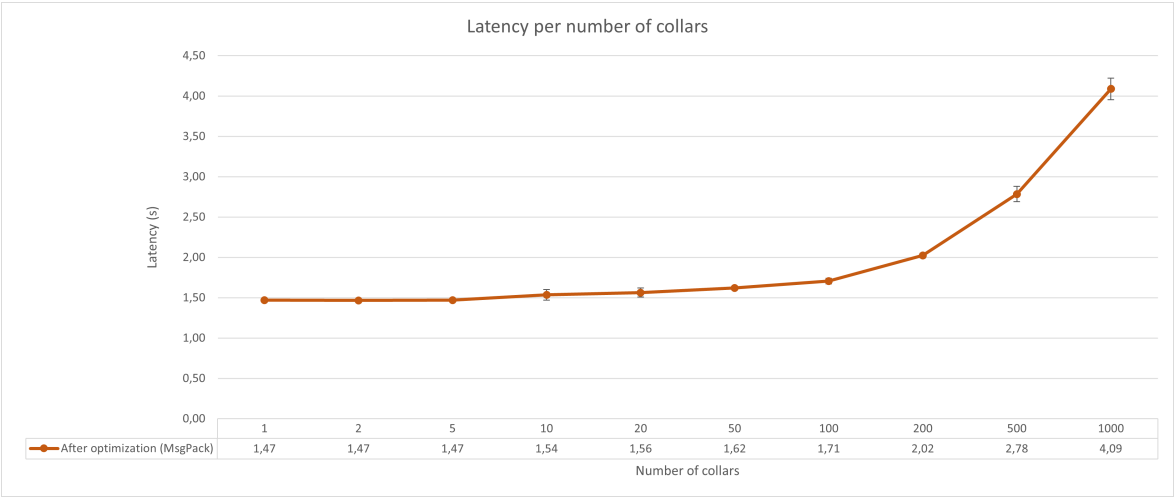


Figure 4.9: Impact of the number of collars on the messages' latency.

4.4 Final system

4.4.1 Setup

After implementing the changes proposed in this dissertation, the final system was tested through the evaluation of the traffic generated and the temporal analysis of the modules modified during this work.

Regarding the first test, the experiment performed analysed the volume of data produced by the final system over 12 days while monitoring a herd. To do so, a dataset containing the alarms generated during that interval was considered. The system was composed by 20 collars and 1 beacon, which did not cover all the pasture area. Moreover, since the animals were not in a vineyard, the conditioning was not enabled, so the devices did not trigger any stimuli upon detecting infractions.

The dataset was analysed and the alarms were grouped by alarm type and by day. The volume of data was calculated using the values of Table 4.2, which were measured in Section 4.2.

Table 4.2: Size of each alarm type.

Device	Alarm Type	Size (Bytes)
Collar	Panic	1159
	Absence	1147
	Battery	1134
	Infraction Shock	1161
	Infraction Buzzer	1126
	Lost	1134
	Equipment	1157
Beacon	Absence	1159
	Battery	1134

As for the second test, the execution time of the modified modules was analyzed, with the system running both on a computer and a raspberry Pi. For each device, the following times were measured:

- Time spent generating alarms: the time it takes from the moment the alarm is detected to when it is stored in the queue.
- Time spent publishing alarms: the time necessary to serialize the alarms and forward them to the CP.
- Time spent publishing WSN information: the time necessary to serialize the information of the WSN and forward it to the CP.

Each test was performed 100 times, the average was considered and the 95% confidence interval was calculated using a z-distribution. A varied number of alarms and collars were simulated, with the values of 1, 20, 100, 500, and 1000.

4.4.2 Analysis

Regarding the first test, Figure 4.10 shows the number of alarms that were generated for the days observed. The measurements of three days were discarded due to the system being switched off. Since most of the alarms are associated with the behavior of living beings, the amount of alarms generated daily does not follow any regular pattern. Furthermore, the number of alarms reached its maximum value on November 24^{th}, when 192 alarms were detected. Approximately 44% of the daily values are in the range of 100 and 140 and 22% of the days generated less than 10 alarms. Moreover, the system registered an average of, approximately, 100 alarms per day.

Table 4.3 displays the number of alarms, grouped by alarm type, that were produced during those days. Some alarm types have greater values than the expected (in a realistic scenario), which is explained by the characteristics of the system in question:

- The infraction alarms were produced despite the conditioning being disabled. Thus, the animals were not warned when committing an undesirable behavior and continued to have the same actions, triggering these alarms repeatedly.
- In the case of the absence alarms, the beacon did not cover the entire pasture area, which means that some animals were not being detected even though they were inside the designated area.

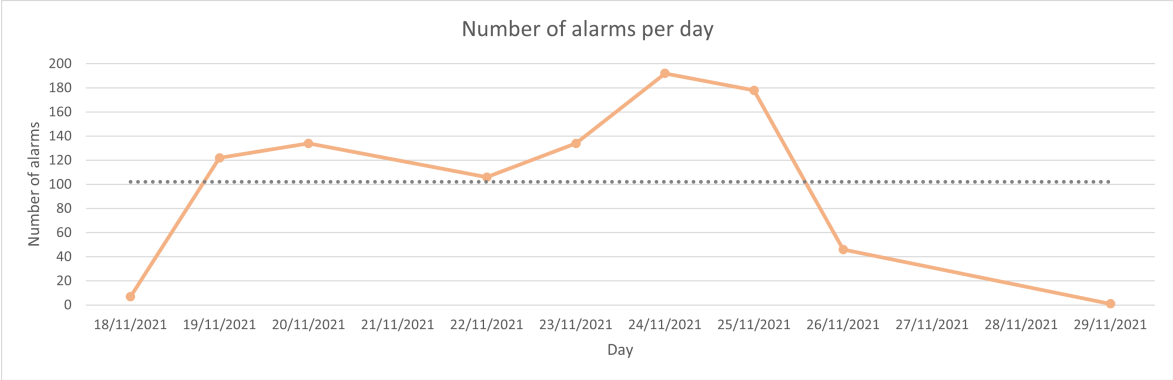


Figure 4.10: Variation of the number of alarms throughout the 12 days.

Considering the size of each alarm type, given in Table 4.2, and the number of alarms produced for each type, presented in Table 4.3, the volume of data produced during the 9 days considered was approximately 1 MB, with an average of 114.2 kB/day. This result shows a great reduction in the daily volume of information generated when compared to the initial system. Likewise, considering the cost of transmission mentioned in Subsection 2.2.1, the daily cost of the system can now be reduced to a value between $114.2 \div 1024 \times 8 \times 4.79 \approx 4$ € and $114.2 \div 1024 \times 8 \times 9.58 \approx 9$ €.

Table 4.3: Number of alarms generated for alarm type.

Device	Alarm Type	Number of alarms
Collar	Panic	6
	Absence	75
	Battery	2
	Infraction Shock	303
	Infraction Buzzer	386
	Lost	0
	Equipment	141
Beacon	Absence	3
	Battery	4

As for the temporal analysis results, the computer presents a greater performance than the Raspberry Pi in every test, which was expected due to its higher processing capacity.

Figure 4.11 and Figure 4.12 show the results for the time measured when generating and publishing the alarms, for three types of alarms. As expected, the execution time increases along with the number of alarms detected.

In the case of Figure 4.11, the results obtained by the raspberry are similar for every type of alarm. However, for the computer, the absence alarm presents a slightly greater execution time for a greater number of alarms, which was not expected, since the function is similar for every type of alarm. This result is likely due to other processes that could be running at the moment and interfered with the test.

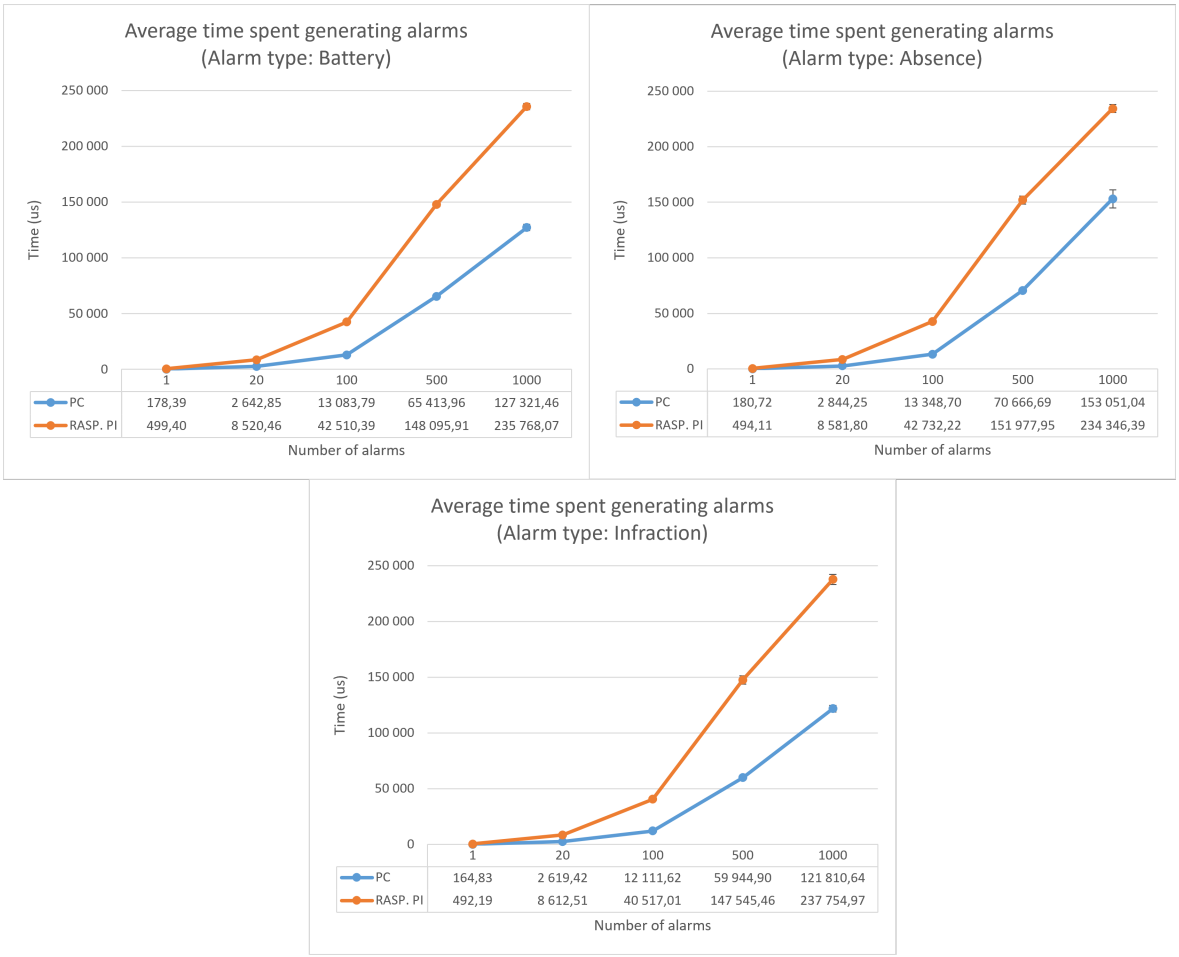


Figure 4.11: Impact of the number of collars on the time spent generating the alarms - for the battery (upper left), absence (upper right) and infraction (bottom) types.

For the Figure 4.12, the results of the raspberry are similar for the battery and absence alarms. In the case of the infraction, the execution time is greater for greater number of alarms. This difference is likely due to the differences on the size of the messages produced by each alarm type. For the computer, the same behaviour was expected. However, the infraction alarm presents lesser execution time from 100 alarms upwards, which can be explained by the interference of some processes in progress during the test of the battery and absence alarms.

Finally, Figure 4.13 shows the results for the publishing of the information of the WSN. As can be seen, the greater the number of collars, the greater is the execution time.

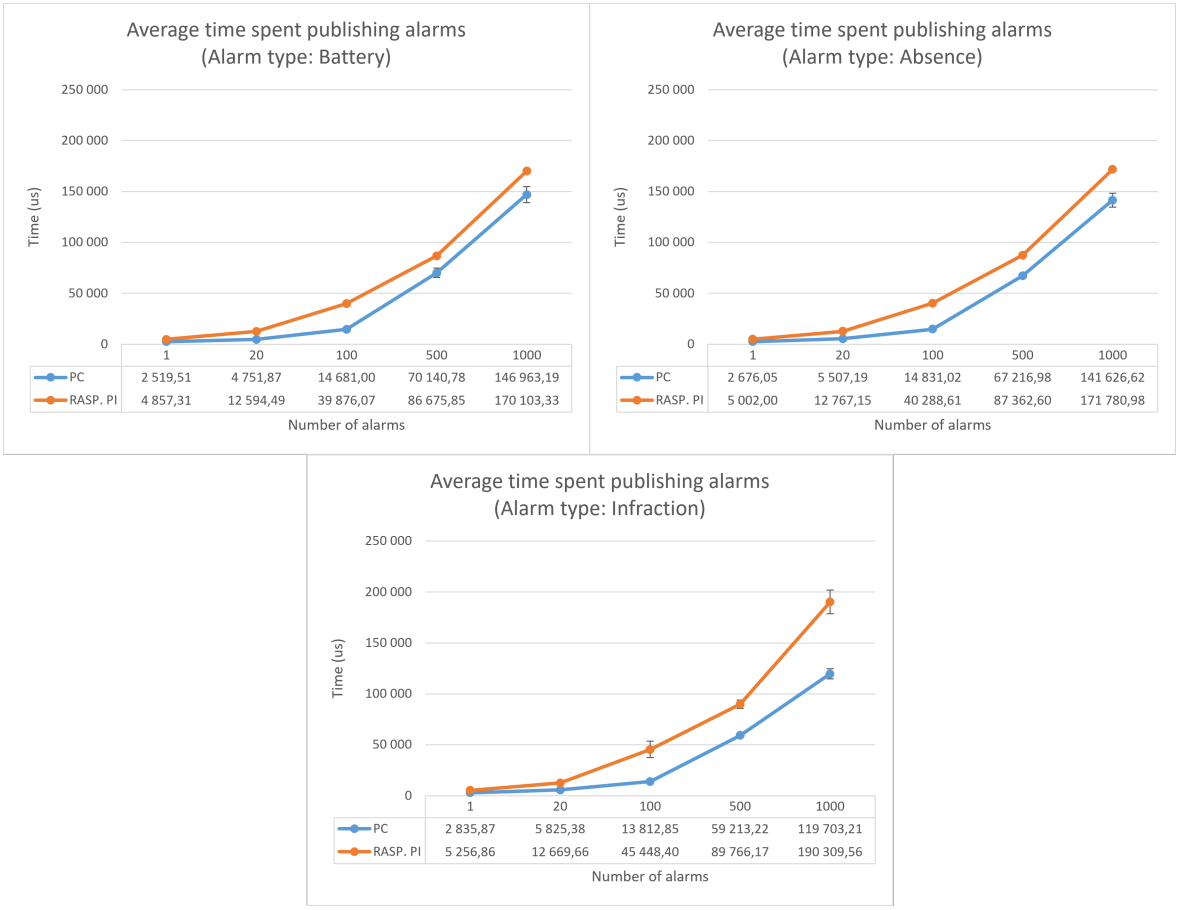


Figure 4.12: Impact of the number of collars on the time spent publishing the alarms - for the battery (upper left), absence (upper right) and infraction (bottom) types.

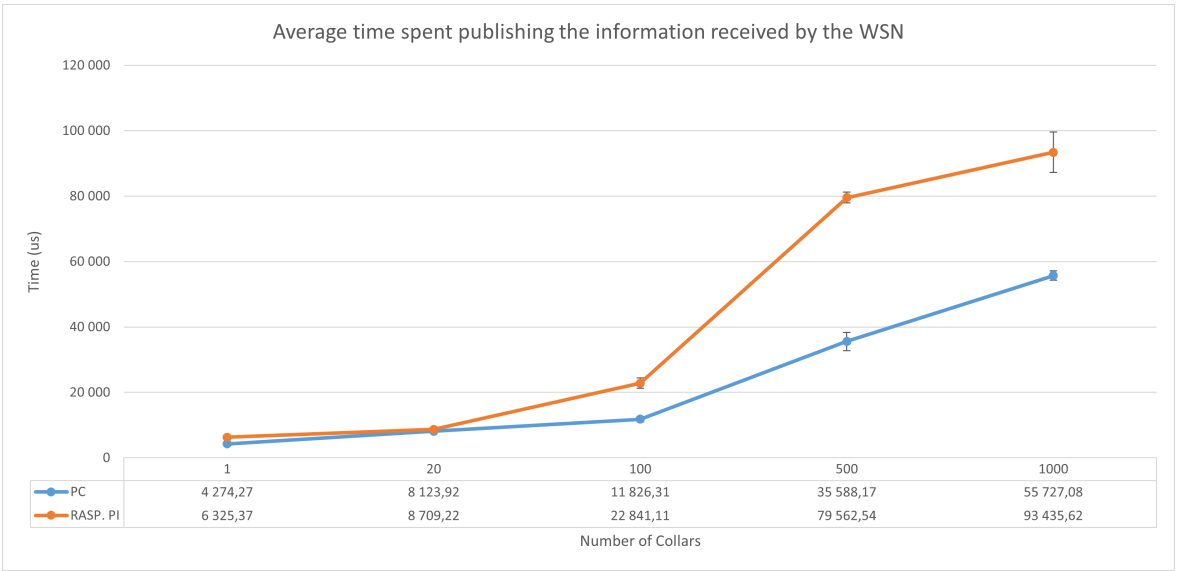


Figure 4.13: Impact of the number of collars on the time spent publishing the information received by the WSN.

CHAPTER 5

Discussion and Conclusion

The use of automated and intelligent systems to support everyday activities is increasing in popularity. Within the agricultural sector, the development of solutions for animal supervision allows shepherds to focus on other agricultural tasks, increasing their productivity. An example of this type of solution is the baseline of this dissertation — the SheepIT project — which provides remote monitoring as well as identifies and alerts for the existence of anomalous situations. Moreover, it allows the use of cattle to control infesting species in vineyards by conditioning their behavior. Similarly to most systems, some of SheepIT's functionalities depend on Internet access. However, most vineyards are situated in rural areas, where the lack of network coverage is common, which hampers the proper operation of SheepIT.

This dissertation aimed to mitigate this gap by extending the system with a satellite communications interface, broadening the scenarios where it could be used. This required various modifications in the system's internal functioning, such as the optimization and replacement of the messages forwarded to the cloud service. For this purpose, the alarm system existing in the gateway was adapted.

Initial testing on the SheepIT solution confirmed the need to reduce the volume of data that was continuously exchanged, so that the requirements of the satellite network could be met. This reduction involved two tasks: the preparation of the alarm system to upload the alarms detected during the animal monitoring, and the optimization of those messages, modifying the encoding strategy that was being used so far. To do so, following some research, three serialization formats were implemented in this work: MessagePack, Protocol Buffers, and Apache Avro. These formats were then compared and the one with the greatest message reduction was chosen. According to the results, MessagePack proved to be the one with the best performance in this context, although Apache Avro presented similar values for alarm types with larger messages. However, the data types chosen to encode the `chars` in the implementation of Apache Avro and Protocol Buffers influenced these results. Regarding the example of Subsection 3.2.3, switching the data type from `bytes` to `int` would result in 1 byte produced instead of 2, which would reduce the payload to 11 and 19 bytes, respectively.

Therefore, Apache Avro would have been the best format, since it requires less bytes to serialize the information.

The system was functionally validated through the evaluation of the volume of information produced as well as its latency during the message exchange. Regarding the first metric, the substitution and optimization of the messages allowed fewer data to be transferred daily. On the other hand, the system experienced higher latency, which is common for a satellite network. Moreover, a temporal analysis was performed to understand the impact of these changes in the gateway's performance. Overall, the results obtained made it possible to verify that the changes performed during this work are perfectly acceptable and compatible with the system, contributing to its improvement. Consequently, the results of this work allowed the submission of a paper for the 10^{th} International Conference on ICT in Agriculture, Food and Environment — HAICTA 2022.

5.1 Future Work

The work described throughout this dissertation allowed the proposed objectives to be achieved. Nevertheless, some aspects can be improved. As previously mentioned, Apache Avro and Protocol Buffers can be further optimized through the modification of some fields' data type. Thus, a new implementation of those formats is suggested as well as the repetition of the comparison tests.

The upload of the alarm information to the CP allowed the creation of a history of alarms, although this information is not accessible to users. This can be resolved by modifying the CP's API to display them. Moreover, it would be interesting to use this data to generate useful statistics, such as which animals usually commit more infractions, panic or get lost more often or which equipment depletes its battery quicker than expected.

Regarding the satellite communications, it would be beneficial to explore alternatives to the EchoStar Mobile network with the purpose of understanding the advantages and disadvantages of using different networks in this context. To do so, it would be useful to evaluate and compare different metrics, such as coverage, latency of messages and data transmission cost.

References

[1] S. Nižetić, P. Šolić, D. López-de-Ipiña González-de-Artaza, and L. Patrono, "Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future," *Journal of Cleaner Production*, vol. 274, 2020, ISSN: 09596526. DOI: 10.1016/j.jclepro.2020.122877.

[2] L. Nóbrega, P. Gonçalves, P. Pedreiras, and J. Pereira, "An IoT-based solution for intelligent farming," *Sensors (Switzerland)*, vol. 19, no. 3, pp. 1–24, 2019, ISSN: 14248220. DOI: 10.3390/s19030603.

[3] A. Temprilho, L. Nobrega, P. Pedreiras, P. Goncalves, and S. Silva, "M2M Communication stack for intelligent farming," *2018 Global Internet of Things Summit, GIoTS 2018*, 2018. DOI: 10.1109/GIOTS.2018.8534560.

[4] L. Nobrega, P. Pedreiras, P. Goncalves, and S. Silva, "Energy efficient design of a pasture sensor network," *Proceedings - 2017 IEEE 5th International Conference on Future Internet of Things and Cloud, FiCloud 2017*, vol. 2017-Janua, pp. 91–98, 2017. DOI: 10.1109/FiCloud.2017.36.

[5] L. J. Ippolito, *Satellite Communications Systems Engineering: Atmospheric Effects, Satellite Link Design and System Performance*. 2008, pp. 1–376, ISBN: 9780470754443. DOI: 10.1002/9780470754443.

[6] O. Kodheli, E. Lagunas, N. Maturo, *et al.*, "Satellite Communications in the New Space Era: A Survey and Future Challenges," *IEEE Communications Surveys and Tutorials*, vol. 23, no. 1, pp. 70–109, 2021, ISSN: 1553877X. DOI: 10.1109/COMST.2020.3028247. arXiv: 2002.08811.

[7] B. R. Elbert, *Introduction to Satellite Communications*. 2008, pp. 1–463, ISBN: 9781630812638.

[8] "Echostar corporation." (), [Online]. Available: https://www.echostarsatelliteservices.com/Satellites# (visited on 02/25/2022).

[9] "Satellite terminals." (), [Online]. Available: https://www.echostarmobile.com/satellite-terminals/ (visited on 02/25/2022).

[10] J. Wei, J. Han, and S. Cao, "Satellite iot edge intelligent computing: A research on architecture," *Electronics (Switzerland)*, vol. 8, no. 11, 2019, ISSN: 20799292. DOI: 10.3390/electronics8111247.

[11] M. De Sanctis, E. Cianca, G. Araniti, I. Bisio, and R. Prasad, "Satellite communications supporting internet of remote things," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 113–123, 2016, ISSN: 23274662. DOI: 10.1109/JIOT.2015.2487046.

[12] N. Islam, M. M. Rashid, F. Pasandideh, B. Ray, S. Moore, and R. Kadel, "A review of applications and communication technologies for internet of things (Iot) and unmanned aerial vehicle (uav) based sustainable smart farming," *Sustainability (Switzerland)*, vol. 13, no. 4, pp. 1–20, 2021, ISSN: 20711050. DOI: 10.3390/su13041821.

[13] S. K. Routray, R. Tengshe, A. Javali, S. Sarkar, L. Sharma, and A. D. Ghosh, "Satellite Based IoT for Mission Critical Applications," *2019 International Conference on Data Science and Communication, IconDSC 2019*, 2019. DOI: 10.1109/IconDSC.2019.8817030.

[14] Z. Qu, G. Zhang, H. Cao, and J. Xie, "LEO Satellite Constellation for Internet of Things," *IEEE Access*, vol. 5, pp. 18 391–18 401, 2017, ISSN: 21693536. DOI: 10.1109/ACCESS.2017.2735988.

[15] B. Petersen, H. Bindner, S. You, and B. Poulsen, "Smart grid serialization comparison: Comparision of serialization for distributed control in the context of the Internet of Things," in *Proceedings of Computing Conference 2017*, vol. 2018-Janua, 2018, pp. 1339–1346, ISBN: 9781509054435. DOI: 10.1109/SAI.2017.8252264.

[16] D. P. Proos and N. Carlsson, "Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV," *IFIP Networking 2020 Conference and Workshops, Networking 2020*, no. i, pp. 10–18, 2020.

[17] A. Sumaray and S. K. Makki, "A comparison of data serialization formats for optimal efficiency on a mobile platform," in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC'12*, 2012, ISBN: 9781450311724. DOI: 10.1145/2184751.2184810.

[18] J. C. Hamerski, A. R. Domingues, F. G. Moraes, and A. Amory, "Evaluating Serialization for a Publish-Subscribe Based Middleware for MPSoCs," *2018 25th IEEE International Conference on Electronics Circuits and Systems, ICECS 2018*, pp. 773–776, 2019. DOI: 10.1109/ICECS.2018.8618003.