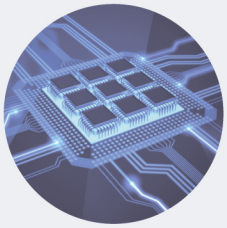
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA DE SISTEMAS COMPUTACIONAIS CRÍTICOS



A WSSL Implementation for Critical Cyber-Physical Systems Applications

MÁRCIA CUNHA ROCHA
outubro de 2023

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

A WSSL Implementation for Critical Cyber-Physical Systems Applications

Marcia Cunha Rocha

Dissertation submitted in partial fulfilment of the requirements for the Master's degree in Critical Computing Systems Engineering

Supervisor: Prof. Dr. Eduardo Manuel Medicis Tovar
Co-Supervisor: Dr. Sergio Duarte Penna

Evaluation Committee:
President:
Luis Miguel Pinho, ISEP

Members:
David Pereira, ISEP
Eduardo Tovar, ISEP

Porto, October 4, 2023

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

The work presented in this document is original and authored by me, and performed in the scope of the Master's degree in Critical Computing Systems Engineering.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration, all references have been acknowledged and fully cited, and all text was originally produced by me (except when duly noted).

I further declare that I have fully followed the Code of Good Practices and Conduct of the Polytechnic Institute of Porto.

Porto, Porto, October 4, 2023

(Signature as in the official identification document, or, preferably, digital signature)

Dedicatory

To my family, especially my mother Erica, who supports me and gives me the strength to persist in all situations, not only during this master's degree.

To my friends who supported me when I moved to Portugal, especially Gleizielly Alves, with whom I shared all the challenges, feelings, and fears of studying in a new country.

To my boyfriend, Michael, who is taking care of me and supporting me in this new period of my life.

To my old teacher and friend Enio Filho, that shared with me his experience, knowledge, and good humor and guided me through the academic world.

To the teachers and researchers at CISTER Research Centre for the assistance, encouragement, and for wisely sharing their knowledge with their students.

To my advisor and co-advisor, Eduardo Tovar and Sérgio Penna, for supporting this work.

Abstract

The advancements in wireless communication technologies have enabled unprecedented pervasiveness and ubiquity of Cyber-Physical Systems (CPS). Such technologies can now empower true Systems-of-Systems (SoS), which cooperate to achieve more complex and efficient functionalities, such as vehicle automation, industry, residential automation, and others. However, for CPS applications to become a reality and fulfill their potential, safety and security must be guaranteed, particularly in critical systems, since they rely heavily on open communication systems, prone to intentional and non-intentional interferences. To address these issues, in this work, we propose designing a Wireless Security and Safety Layer (WSSL) architecture to be implemented in critical CPS applications. WSSL increases the reliability of these critical communications by enabling the detection of communication errors. Otherwise, it increases the CPS security using a message signature process that uniquely identifies the sender. So, this work intends to present the WSSL architecture and its implementation over two different scenarios: over Message Queue Telemetry Transport (MQTT) protocol and inside a simulation environment for communication between Unmanned Aerial Vehicles (UAVs) and Ground Control Stations in case of Beyond Visual Line of Sight (BVLOS) applications. We aim to prove that the WSSL does not significantly increase the system payload and demonstrate its safety and security resources, allowing it to be used in any general or critical CPS.

Keywords: Safety, Security, Cyber-Physical Systems, Critical Systems

Resumo

Os avanços nas tecnologias de comunicação sem fios permitiram uma omnipresença e ubiquidade sem precedentes dos Sistemas Ciber-Físicos (CPS). CPS são a combinação de um sistema físico, um sistema cibernético, e a sua rede de comunicação. Tais tecnologias podem agora capacitar verdadeiros Sistemas de Sistemas (SoS) que cooperam para alcançar funcionalidades mais complexas e eficientes, tais como automação de veículos, indústria, automação residencial, e outras. As aplicações CPS são baseadas num ambiente complexo, onde sistemas estão interligados e dispositivos interagem entre si em grande escala. Estas circunstâncias aumentam a superfície de ataque, e os desafios para garantir fiabilidade e segurança. Contudo, para que as aplicações CPS se tornem realidade e alcancem o seu potencial, a segurança do funcionamento e segurança contra intrusões devem ser garantidas, particularmente em sistemas críticos, uma vez que dependem fortemente de sistemas de comunicação abertos, propensos a interferências intencionais e não intencionais. Tais interferências podem ocasionar graves danos ao ambiente e riscos a integridade física e moral das pessoas envolvidas. Neste trabalho, propõe-se a concepção de uma arquitectura WSSL, a ser implementada em aplicações críticas de CPS, para abordar estas questões. Esta arquitectura aumenta a fiabilidade das comunicações críticas, permitindo a detecção de erros de comunicação. Além disso, aumenta a segurança dos CPS utilizando um processo de assinatura de mensagem que identifica de forma única o remetente, garantindo a integridade e autenticidade, pilares cruciais da cibersegurança. Assim, pretende-se apresentar a definição, arquitectura e a implementação da WSSL sobre um protocolo MQTT (do inglês Message Queue Telemetry Transport) para avaliação dos custos associados a sua implementação, e provar que esta não aumenta significativamente a carga útil do sistema. Também é pretendido avaliar seu comportamento e custos a partir da implementação em um ambiente simulado para comunicação entre veículos aéreos não tripulados e estações de controle terrestres . Por fim, deve-se avaliar se os seus recursos de segurança são eficientes na detecção de erros relativos a segurança do funcionamento ou a segurança contra intrusões, permitindo a sua utilização em qualquer CPS, seja ele um CPS crítico ou não.

Acknowledgement

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID-P/UIDB/04234/2020); by the FCT and the Portuguese National Innovation Agency (ANI), under the CMU Portugal partnership, through the European Regional Development Fund (ERDF) of the Operational Competitiveness Programme and Internationalization (COMPETE 2020), under the PT2020 Partnership Agreement, within project FLOYD (grant nr. 45912, POCI-01-0247-FEDER-045912); by FCT and the EU ECSEL JU under the H2020 Framework Programme, within project ECSEL/0010/2019, JU grant nr. 876019 (ADACORSA); also by project FLY-PT (grant nr. 46079, POCI-01-0247-FEDER-046079), co-financed by the European Regional Development Fund (ERDF) within COMPETE 2020, in the scope of PORTUGAL 2020. The JU receives support from the European Union's Horizon 2020 research and innovation program and Germany, Netherlands, Austria, France, Sweden, Cyprus, Greece, Lithuania, Portugal, Italy, Finland, and Turkey. The ECSEL JU and the European Commission are not responsible for the content of this paper or any use that may be made of the information it contains.

Contents

List of Figures **xv**

List of Tables **xvii**

List of Source Code **xix**

List of Acronyms **xxi**

1 Introduction **1**
1.1 Motivations and Challenges . . . 1
1.2 Contributions . . . 3
1.3 Outline . . . 3

2 State of the art **5**
2.1 Cyber Physical Systems . . . 5
2.2 CPS Applications and Challenges . . . 6
2.3 Security and Safety . . . 8
2.4 Cybersecurity . . . 10
2.5 Communication Protocols for CPS . . . 11
2.5.1 Message Queue Telemetry Transport . . . 12

3 Development **13**
3.1 WSSL Definition . . . 13
3.2 WSSL Architecture . . . 15
3.2.1 Signature Method . . . 16
3.3 WSSL Components . . . 19
3.3.1 WSSL Sender . . . 22
WSSL Sender entity for safety . . . 22
WSSL Sender entity for security . . . 23
3.3.2 WSSL Receiver . . . 25
WSSL Receiver entity for security . . . 26
WSSL Receiver entity for safety . . . 27

4 Evaluation **31**
4.1 Methodology . . . 31
4.2 Evaluation using MQTT . . . 32
4.2.1 WSSL Cost . . . 34
4.3 Inter-message delay . . . 35
4.3.1 Error detection . . . 35
4.4 Tests Results and Conclusions . . . 37

5 Applications **41**
5.1 WSSL and CopaDrive 41
5.2 WSSL and ADACORSA 43
5.2.1 WSSL Cost 44
5.2.2 Error detection 45
5.3 Tests Results and Conclusions 47

6 Conclusions **51**

Bibliography **53**

7 Appendix A - WSSL Instalation Tutorial **59**
7.1 How do I get set up? 59
7.1.1 Make and CMake 59
7.1.2 CryptoIdentity library 60
7.1.3 Using WSSL with MQTT Libmosquitto 60

List of Figures

2.1 Ground Station simulation using QGroundControl. 7

3.1 Basic implementation of WSSL. 13
3.2 WSSL covered threats and defense methods. 14
3.3 General flow of the WSSL Library. 15
3.4 Illustration of sending three equal messages. 16
3.5 Sender and Receiver files containing the Identities information. 18
3.6 WSSL Sender's safety. 23
3.7 WSSL Sender's security. 24
3.8 WSSL Sender sequence diagram. 25
3.9 WSSL Receiver's security. 26
3.10 WSSL Receiver's safety. 27
3.11 Receiver sequence diagram. 29

4.1 Laboratory using MQTT and WSSL 31
4.2 WSSL's delay detection - MQTT. 35
4.3 Log file of generated errors. 36
4.4 WSSL average costs versus frequency and number of messages when integrated with MQTT. 37
4.5 WSSL's safety entity costs in percentage versus frequency and number of messages. 38
4.6 WSSL's security entity costs in percentage versus frequency and number of messages. 39
4.7 WSSL costs in percentage versus frequency and number of messages. . . . 39

5.1 CopaDrive Architecture (Filho, Severino, Rodrigues, et al. 2021). 42
5.2 How WSSL was planned to be integrated with CopaDrive. 42
5.3 Laboratory setup for evaluating WSSL in the Handover code. 44
5.4 WSSL's costs in percentage for different frequencies - ADACORSA. 45
5.5 Telemetry velocities without WSSL. 46
5.6 Telemetry velocities with WSSL. 46
5.7 Sequence diagram illustrating the communication between Drone and GCS using WSSL. 47
5.8 WSSL average costs versus frequency and number of messages when integrated with ADACORSA. 48
5.9 WSSL's inter-message delay detection in ADACORSA. 49

List of Tables

4.1 Sent and reception time costs when sending fifty thousand msgs with the frequency of 1000 Hz. 34

5.1 Sent and reception time costs when sending fifty thousand msgs with the frequency of 1000 Hz and 333 Hz. 48

List of Source Code

3.1 Fuction responsible to create the Identifier (ID) file. 17
3.2 Fuction responsible to a new identity in the known identity list of the ID. . 17
3.3 Signing the message coming from safety. 18
3.4 Verifying the signature and recovering the message coming from the Sender Device (SD). 19
3.5 Return classes of the Sender and Receiver. 19
3.6 Packet format of the Sender and Receiver tables. 20
3.7 Init function for the WSSL_Sender. 20
3.8 Init function for the WSSL_Receiver. 20
3.9 Functions responsible for deleting old connexions in the table. 21
3.10 Functions responsible for deleting old connexions in the table. 21
3.11 Public variables inside the WSSL class. 21
3.12 Treating the invalid argument error. 21
3.13 Different ways to instantiate the WSSL Sender. 22
3.14 Different ways to instantiate the WSSL Receiver. 25
4.1 Function publishing in topic "wssl" in Mosquitto broker. 32
4.2 Function subscribed in topic "wssl" in Mosquitto broker. 33
5.1 Definition of the drone telemetry struct. 43

List of Acronyms

ADACORSA	Airborne Data Collection on Resilient System Architectures.
AMQP	Advanced Message Queuing Protocol.
BVLOS	Beyond Visual Line of Sight.
CAGR	Compound Annual Growth Rate.
CAM	Cooperative Awareness Message.
Co-CPS	Cooperative Cyber-Physical Systems.
Co-VP	Cooperative Vehicular Platooning.
CoAP	Constrained Application Protocol.
CPS	Cyber-Physical Systems.
DDS	Data Distribution Service.
ERTMS	European Traffic Management System.
ETSI	European Telecommunications Standards Institute.
GCS	Ground Control Station.
HTTP	HyperText Transfer Protocol.
ID	Identifier.
IoT	Internet of Things.
ITS	Intelligent Transport System.
M2M	Machine-to-Machine.
MQTT	Message Queue Telemetry Transport.
pk	public key.
QoS	Quality of Service.
RD	Receiver Device.
ROS	Robot Operating System.
SD	Sender Device.
sk	secret key.
SoS	Systems-of-Systems.

UAVs	Unmanned Aerial Vehicles.
V2I	Vehicle-to-Infrastructure.
V2V	Vehicle-to-Vehicle.
V2X	Vehicle-to-Everything.
WSSL	Wireless Security and Safety Layer.

Chapter 1

Introduction

Cyber-Physical Systems (CPS) applications are gaining prominence due to advances in wireless communications in several areas, including industry (Hermann, Pentek, and Otto 2016), logistics (Coopmans et al. 2015), smart buildings (Pivoto et al. 2021) and vehicle automation (Rawat and Bajracharya 2017). These systems offer many benefits, like removing people exposition to dangerous conditions, improving safety, reducing costs, and increasing flexibility and mobility. Moreover, such technologies can now empower true Systems-of-Systems (SoS), which cooperate to achieve more complex and efficient functionalities, enabling various services and applications and involving several interconnected systems.

Nevertheless, CPS require additional attention for critical environments where malicious activities or network issues can greatly damage or put lives at risk. Many CPS devices, whether cooperative or not, actuating in open communication environments, are mainly subject to malicious agents. Furthermore, their implementation based on devices from different manufacturers allows the possibility of security flaws (*EN 50159* 2010).

The criticality of CPS also arises from the fact that these systems heavily rely on wireless communications to exchange safety-critical information (Vieira et al. 2019). Hence, the *wireless communication* is commonly susceptible to unauthorized access due to its inherent open transmission system and broadcast nature (J. Zhang et al. 2017). In addition, the amount of systems relying on wireless communication to transmit messages is increasing, contributing to an extension of the attack surface. Therefore, for CPS applications to become a reality and fulfill their potential safety and security must be guaranteed.

Research on safety and security for CPS devices is broad and widespread in the literature, but only a few address security and safety in a practical way. According to (Kavallieratos, Katsikas, and Gkioulos 2020), between sixty-eight analyzed methods for cyber-security and safety co-engineering, less than half are aware of security and safety standards or even include information on the validation of the method they propose. The authors also state that [...]" the applicability to different application domains is usually not demonstrated in most reviewed methods," and several important issues remain open.

1.1 Motivations and Challenges

Many well-known international companies operate in the CPS market, including IBM Corporation, Microsoft, Siemens, Astri, MathWorks, and Intel. Additionally, market size projections show a large potential for the CPS market. The global CPS market is estimated to reach 137 billion dollars by 2028 with a projected Compound Annual Growth Rate (CAGR), which measures the return on investment over time, of 8.2 percent over the forecast period

between 2022 and 2030 (*Analytics Market Research* 2023). It also underlines that the CPS market represents around 40 percent of the global market for information security.

According to the United Kingdom government's Cyber Security Breaches Survey 2022 (Ell and Gallucci 2022), 39 percent of the businesses identified a cyber-attack in the last 12 months of the survey period. Similarly, cybercrime research in the United States concluded that [...] " over 53 million individuals were affected by data compromises, which include data breaches, data leakage, and data exposure" (Statista 2022) in the last half of 2022. Moreover, Gartner, Inc. (Moore 2020) predicts that the financial impact of CPS attacks resulting in fatal casualties will reach 50 billion dollars in compensation, litigation, insurance, and reputation loss in 2023.

CPS applications are based on a complex environment with devices interacting with each other on a large scale and interconnection among systems. These circumstances increase attack surfaces, bringing new challenges regarding reliability and security. Simultaneously, adding safety and security mechanisms to protect these applications may increase the system's complexity and transmission costs. As a result, methods and tools have been developed to deal with CPS complexity, such as approaches involving separating the system into parts to facilitate their management, modeling formalisms, and tools for verifying and validating software and systems (Navet and Merz 2013).

According to (Törngren and Sellgren 2018), some key elements to deal with complexity for CPS are decision-making and developing means to improve the management of uncertainty and risk during the design stage, for instance, using software security certification to build on existing software assurance, validation, and verification techniques. However, the authors also state that these practices are often weak, and integrating such systems is usually identified as a challenge that consumes time and increases costs. On the other hand, certification cannot provide proof of the status of a system dynamically (Munoz and Mafia 2014), which is a must for heterogeneous and unpredictable systems such as CPS.

According to (Baheti and Gill 2019), these certification and formal modeling approaches are insufficient for verifying the safety and correctness of designs at the system level and component-to-component physical and behavioral interactions. The author also states that certifications at the control design stage consume more than 50 percent of the resources needed to develop new safety critical systems in several application domains, such as aviation, automotive, medical, and energy systems. Therefore, there is an urgent need for standardized abstractions and architectures that allow modular design and development of CPS.

This thesis aims to fill the gap of a modular, low-cost, open-source application that addresses unified safety and security features for CPS. So, it fulfills the necessity of a generic and portable application that can be applied between all kinds of CPS without significantly increasing the communication costs, system complexity, and network overhead.

This work is inserted in the FLY-PT project (FLY-PT 2023), which will develop the prototype of a personal air transport system at scale. This prototype will consist of an autonomous aerial vehicle (drone), allowing air mobility, an autonomous vehicle, allowing land mobility, and a cabin that can be attached to each of the two vehicles.

1.2 Contributions

This thesis will present a modular Wireless Security and Safety Layer (WSSL) architecture, establishing a safe way to exchange information in CPS. The proposed WSSL does not rely on standard transmission systems such as gateways and protocols, applying to a wide range of applications that demand secure transmissions and safe applications, including simulated and real environments (Filho, Guedes, et al. 2020). Moreover, although some defenses involve verifying the origin and destination of the messages sent, the WSSL is agnostic to the message contents or application payload, guaranteeing the data's trust and privacy. In addition, its implementation is independent, as much as possible, of the communication stack used. Thus, the contributions of this work can be divided into the following main aspects:

- Demonstrate the architecture of WSSL, introducing its concept and agnostic model, based on a black channel modeling, for communication in insecure media.
- Evaluate the implementation of WSSL using Message Queue Telemetry Transport (MQTT) as the communication protocol between two devices, measuring the impact of its use on the data network through quantitative and qualitative analysis.
- Evaluate the implementation of WSSL inside a project that aims Beyond Visual Line of Sight (BVLOS) communication Handover, ADACORSA (ADACORSA 2023), measuring the impact of the integration through quantitative and qualitative analysis.
- Demonstrate the ability of WSSL to add safety features to the message: Sequence Number, Identifier (ID), and Timestamp.
- Demonstrate the ability of WSSL to add security features (integrity and authenticity) to the message using a digital signature.
- Demonstrate the ability of WSSL to use its features to detect attack actions on the CPS, monitor network problems, and report them to the application, increasing the application's security and safety.
- Writing and publishing of an ACM/IEEE conference article (Cunha Rocha et al. 2023).

1.3 Outline

This work is divided as follows: Chapter 2 represents the state of the art, presenting background information regarding the different components of this thesis and explaining some essential concepts to understand the WSSL environment. At the same time, it has an overview of the related works and an analysis of the proposed problem. Chapter 3 describes the concepts and design of the WSSL algorithm and how its components work together to improve safety and security. Chapter 4 details the evaluation methodology, namely, how the tests were developed, the achieved results, and the gathered conclusions. Chapter 5 presents the application of WSSL in the telemetry of an uncontrolled environment within Unmanned Aerial Vehicles (UAVs) and a Ground Control Station (GCS). Moreover, the evaluation of the integration in this system is described, and the results are analyzed. At last, 6 concludes this thesis, headlining the principal results and contributions and promoting possible directions for future works.

Chapter 2

State of the art

This chapter introduces a contextualization and a good overview of the state-of-the-art. Section 2.1 presents the CPS, its context, and its characteristics. Then, the theory for CPS safety and security and the standards related to this work are described in section 2.3. Moreover, the CPS cybersecurity threats are briefly presented in section 2.4 to clarify the defenses WSSL aims to implement. At last, the communication protocols used in CPS are presented in section 2.5, together with the general specifications of the MQTT, the protocol intended to be used during the evaluation tests.

2.1 Cyber Physical Systems

Cyber-Physical Systems (CPS) address a new generation of systems that integrates cyber and physical components, that is, with computational and physical capabilities that can interact with humans through several modalities. "CPS are spatially-distributed, time-sensitive, and multi-scale, networked embedded systems [...]" (Esterle and Grosu 2016) that can connect to the physical environment using modern sensors and actuators with computational capabilities and network technologies.

CPS allows combining technologies and knowledge and has been widely adopted (Alguliyev, Imamverdiyev, and Sukhostat 2018). According to (Bilenko et al. 2020), CPS are key to ensuring competitiveness in companies engaged in industry and manufacturing processes. It can bring several advantages for new systems, such as autonomy, reliability, and control, without the need for human intervention. "The ability to interact with, and expand the capabilities of, the physical world through computation, communication, and control is a key enabler for future technology developments" (Baheti and Gill 2019).

Often related to *Industry 4.0* and *Internet of Things (IoT)*, CPS are the basis for the development of *smart technologies* such as smart manufacturing, smart infrastructures, smart city, smart vehicles, and others. They offer many benefits to modern society (Esterle and Grosu 2016), like removing people exposition to dangerous conditions by reducing traffic fatalities, improving safety and control in smart cities, reducing costs through resource conservation, facilitating network management, and increasing flexibility, autonomy, and mobility.

In Industry 4.0, also known as Smart Industry, CPS connects machines, products, data, and service providers, allowing new ways of organizing and conducting industrial processes and ensuring an appropriate interaction between the areas (Hermann, Pentek, and Otto 2016). In the Smart Building, the interaction between CPS and smart devices can reduce energy consumption and increase the resident's protection, safety, and comfort (Pivoto et al. 2021).

In smart vehicles, CPS combines computation, communication, and control between vehicles and vehicle and infrastructure. As a result, they improve road safety, efficiency, comfort, and quality of life by reducing traffic congestion, accidents, and fuel consumption (Rawat and Bajracharya 2017).

There is an increasing consensus about the similarities between the CPS and IoT concepts. According to (Greer et al. 2019), both comprise interacting logical, physical, transducer, and human components and have overlapping definitions. However, the author highlights that they emerge from different communities. CPS primarily emerges from a system engineering and control perspective. In contrast, IoT emerges from a networking and information technology perspective. So, CPS are interconnected systems collaborating through the IoT systems (Marwedel 2021), and the IoT can be considered the backbone of CPS (Esterle and Grosu 2016). Thus, CPS came to integrate physical and computing elements (Wolf and Serpanos 2018), which unlocked a wide range of potential cyber-derived threads.

2.2 CPS Applications and Challenges

Critical systems applications such as the drone and automotive industries brought different challenges to the developing of CPS. For example, drone operations must be safe, reliable, and secure in all situations and flight phases. In this scenario, Beyond Visual Line of Sight (BVLOS) technologies enhance the autonomy of drones, and their use goes beyond military applications. Furthermore, BVLOS allows complex interactions between Unmanned Aerial Vehicles (UAVs), empowering the integration in aerial photography, search and rescue, commercial delivery, infrastructure inspection, and surveillance (Politi et al. 2022).

These applications represent a new class of CPS, the Cooperative Cyber-Physical Systems (Co-CPS), and have many benefits and significant economic potential in numerous domains (Kabir 2021). Likewise, they bring a new paradigm for these systems, increasing their capacity to perform critical activities with real-time constraints. Nevertheless, several challenges emerge with Co-CPS, raising concerns regarding the reliability and security of communications and their impact on safety and efficiency in the environment where such systems are implemented.

UAVs, known as Drones, confront several safety and security issues as they are planned to be deeply inserted in society's safety-critical activities. As it is broadly discussed nowadays, Drone networks are vulnerable to many types of privacy and security threats. The connections between drones are also vulnerable since they rely on mobile wireless networks and may experience challenges due to limited power, high movement speed, packet loss, network congestion, and others (Shayea et al. 2022).

Ensuring reliable, smooth, and continuous connectivity for Drones is one of the major challenges in their implementation. More problems emerge from the fact drones depend on frequent handovers. **Handover** is a method to maintain continuous active sessions of the users during base station or sector switching connection (Gódor et al. 2015). It is a crucial mechanism for wireless communication systems and even more important when speaking about UAVs operating as BVLOS.

Handovers are naturally exposed to cyber-attacks. Moreover, erroneous transmissions from limited coverage areas in drone networks can disrupt the service. These problems and many others increase the need for securing critical infrastructures and developing new safety and

security solutions for UAVs, which made some European and international collaborations formed among industrial and academic partners.

When working with critical systems, research and development of components and systems are necessary to reduce risk, time, and costs. With that in mind, many important projects have been developed to address Co-CPS challenges. For instance, Airborne Data Collection on Resilient System Architectures (ADACORSA) is an EU-funded project aiming to reinforce the drone industry and increase public and regulatory acceptance of BVLOS technologies by demonstrating technologies for safe, reliable, and secure drone flight in all circumstances and phases. In addition, ADACORSA covers the development of an authority Handover between Ground Control Station (GCS) and Drones without human interference, and the future integration with WSSL represents an essential part of the project.

One of the ADACORSA's software uses Ardupilot firmware in the drone and the QGroundControl simulator for communication and simulation of the Ground Control Stations. The ArduPilot Project (ArduPilot 2023) is an open-source project that enables the creation and use of trusted, autonomous, unmanned vehicle systems. It provides an advanced, full-featured, and reliable autopilot software system and a comprehensive suite of tools suitable for almost any vehicle and application.

The QGroundControl (QGroundControl 2023) provides full flight control and vehicle setup for PX4 or ArduPilot-powered vehicles. Its primary goal is to facilitate the use of professional users and developers, providing straightforward usage for beginners while delivering high-end feature support for experienced users. Moreover, QGroundControl is open-source and provides full flight control and mission planning for any MAVLink-enabled drone. Figure 2.1 shows the GroundControl simulating the drone flying a configured mission.

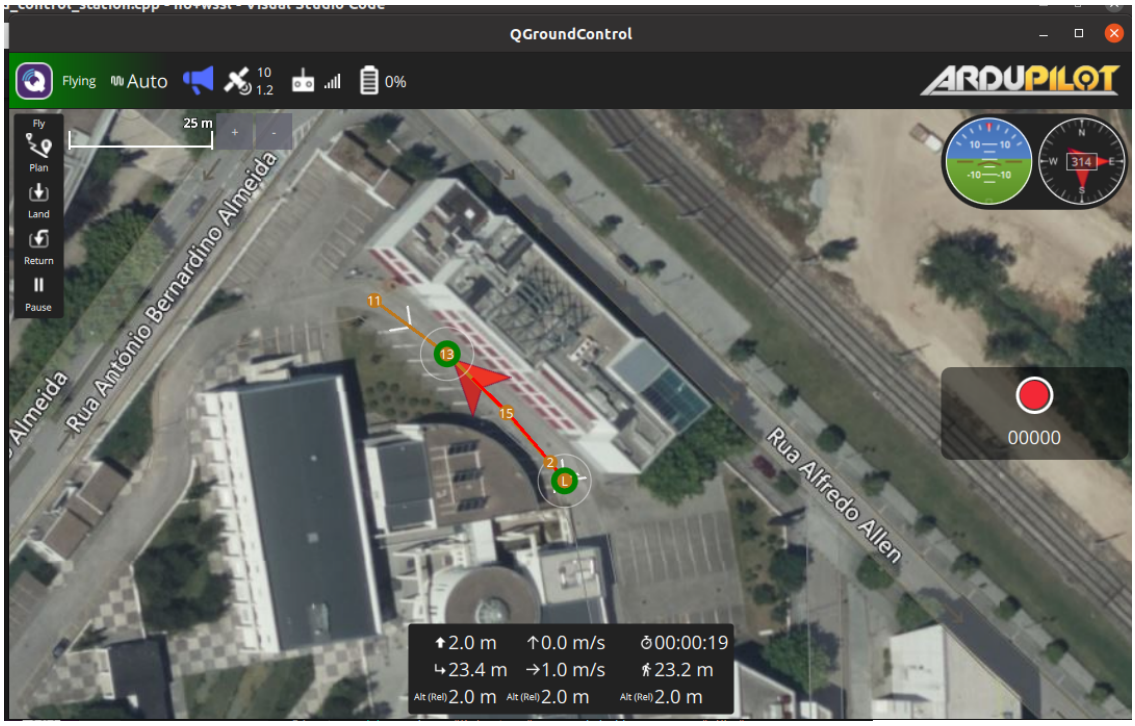


Figure 2.1: Ground Station simulation using QGroundControl.

There is still much work to do regarding developing and validating Drone capabilities and specific services and procedures to ensure secure operation and traffic management, allowing

efficient and safe access to airspace for a substantial quantity of Drones. As pointed out, UAVs require efficient mobility, safe handover techniques, and a continuous, stable, and reliable network connection. With that in mind, Integrating WSSL in such a project would allow service providers to perform complex drone operations safely and increase the Handover security.

Accordingly, WSSL will be implemented as extra protection for the telemetry data of UAVs. Drone telemetry is used to provide drone position, velocity, and timing (PVT) information to the operator along with other information such as heading, battery status, distance to home, flight time, attitude, and others (*European Global Navigation Satellite Systems (EGNSS) for drones operations: white paper* 2020).

On the other hand, autonomous driving has achieved unprecedented growth, and connectivity between vehicles (V2V) and between vehicles and infrastructure (V2I) enabled various advanced applications. In this scenario, a Cooperative Vehicular Platooning (Co-VP) is vital to advancing the safety and efficiency of autonomous driving by increasing road capacity and fuel efficiency. Thus, CopaDrive (Filho, Severino, Rodrigues, et al. 2021) emerges as a simulation tool that evaluates the security and reliability of a Co-VP system and whose WSSL would add a powerful resource to increase its capability to detect network problems.

CopaDrive is a 3D vehicle simulation tool integrated with an European Telecommunications Standards Institute (ETSI) standard (ETSI EN 302 663 2012) operating in the 5 GHz frequency band, known as Intelligent Transport Systems (ITS-G5), and a network simulator OMNET++, which evaluates network problems, such as delays and packet losses, on the security and reliability of a Co-VP system. CopaDrive works with Robot Operating System (ROS), a framework that facilitates the development process of autonomous vehicles by providing multiple libraries, tools, and algorithms (*ROS - Robot Operating System* 2022). ROS is an integrative tool since it also supports several devices to simulate the physical environment and many sensors and actuators for CPS.

Through a 3D simulator, Gazebo, and OMNET++, CopaDrive is an integrated framework for developing, testing and evaluating Co-VP systems in different environments. For example, it assesses how the communication between vehicles impacts the ability of the platoon to perform safely and how network problems, such as delays and packet losses, can significantly impact the security and reliability of the Co-VP system.

CopaDrive defined an architecture that would minimize the implementation effort in real platforms. Thus, WSSL emerges as a tool to increase safety and security in CopaDrive. Furthermore, it can be integrated with minimal resources and effort, optimizing its operation and improving its safety and security features. Integrating both works would allow a new paradigm with new methods for evaluating WSSL efficiency, advantages, and possible limitations. In future works, it could also be extended for real scenarios, using ROS capabilities to expand to implement a hybrid environment.

2.3 Security and Safety

The safety standard EN 50159 (*EN 50159* 2010) helps to classify transmission systems into three categories. Category 1 comprises systems under the designer's control and fixed during their lifetime. Category 2 of systems partly unknown or not fixed, but unauthorized access can be excluded, and Category 3, where systems are not under the control of the designer, and unauthorized access has to be considered. Category 1 and 2 type communication

systems are often referred to as **closed transmission** while Category 3 is referred to as **open transmission**.

Although an open transmission system always has the potential for unauthorized access, wireless communication is still the favorite option to avoid installation costs and enable the connection between CPS devices in an IoT system, increasing the systems' mobility. So, as CPS will generally contain sensitive, private, or confidential information, it must dispose of methods to guarantee safety and security, ensuring the data authenticity, integrity, and availability at the receiving end.

Accordingly, EN 50159 proposes an end-to-end safety approach using the black channel principle (Filho, Severino, Koubaa, et al. 2021). In this approach (*IEC 61508* 2010), safe applications are implemented over a non-secure transmission system with non-certified network communication. Therefore, the transmission system is considered unsafe, and safety and security mechanisms are implemented as separate layers in each end node in the communication. The safety layer is in charge of the defenses against random and systematic faults and failures, and the security layer protects against deliberate threats from external sources with malicious purposes.

It is important to define the terminologies of safety and security and their differences and interdependencies to clarify this thesis's scope, objective, and motivations.

Safety and security have a variety of definitions depending on the context and communities (Esterle and Grosu 2016; Lyu, Ding, and Yang 2019; Wolf and Serpanos 2018). Both are associated with risks and can cause different consequences (e.g., human losses, environmental damage, financial loss), being differentiated in the risk source. Under the environment covered by this thesis, *safety* considers hazards with potential impact on the system environment. At the same time, *security* focuses on threats that could impact the environment and the system itself. Thus, *safety* can be defined as accidental risks originating from the system. *Security* is related to malicious threats caused by intentional attacks that may impact a vulnerable system and its operation.

Traditionally, much attention has been given to system safety, such as accidental component failures and software errors. Over the years, safety standards have been created to standardize and ensure safety for different applications. For example, IEC 61508 (*IEC 61508* 2010) is a safety standard that addresses functional safety for electrical, electronic, and programmable electronic systems. This norm defines *safety* as freedom from unacceptable risk of harm to humans, directly through injury or death, or indirectly due to damage to equipment, property, or the environment.

Safety standards like IEC 61508 define several aspects and models to increase systems safety but still need to advance to real applications (Magro, Pinceti, and Rocca 2016). Furthermore, most works encompassing safety are closed in a specific application field, reducing their utilization to general CPS. For instance, EURORADIO (Cecchetti et al. 2013; L.-j. Chen et al. 2011) is a safety layer related to railways control systems, defined by the European Traffic Management System (ERTMS). WirelessHART is a standard that addresses the main concerns about reliability, security, and integration for real-world industrial applications (Akerberg et al. 2011), and ISO 26262 and SAE J3061 define international vehicle safety and security standards used for autonomous vehicles (Schmittner et al. 2016). Such complex solutions related to CPS are not easily portable to other scenarios. Thus, current safety measures are inadequate and intrusive, and many research proposals still need to be practical and cost-effective.

Furthermore, safety and security have been discussed independently by distinct communities in the literature. Several kinds of research focus on the CPS security issues, suggesting the use of cryptography (J. Zhang et al. 2017), attack detection methods (D. Zhang et al. 2021), and other security mechanisms (Lind 2020; Xiao, H.-H. Chen, et al. 2006; Xiao, Sethi, et al. 2005). Just a few studies comprise safety and usually are focused on a theoretical approach (Balador et al. 2018), like surveys analyzing existing methods (Hoffmann et al. 2018) or suggesting the importance of safety in CPS devices (Atlam and Wills 2020). On the other hand, some complex systems are being developed: a risk assessment framework focused on quick and guided feedback about safety and security (Asplund et al. 2019), or a cyber-physical system analytical framework (Vinel, Lyamin, and Isachenkov 2018), where, despite their importance, neither of them are easily implemented into the system or provide operational tools that simplify their integration.

CPS are often safety-critical systems (Nandi, Pereira, and Proença 2021). Integrating software components across safety-critical infrastructures is sensitive to a wide range of security attacks, bugs, and design flaws, and their failure can cause grave consequences (Johnson 2012). Several studies raised concerns about new security threats and their potential impact on safety, concluding that the security and safety of CPS often influence each other (Carreras Guzman, Kozine, and Lundteigen 2021; Ji et al. 2021; Kriaa et al. 2015; Lyu, Ding, and Yang 2019; Zhou et al. 2021). Still, only a few address both of them in a practical way, providing adequate methods, theory, and tools to cover CPS complexity. Thus, to bridge the gap concerning these limitations, CPS must be designed and operated under a unified view of safety and security characteristics.

Therefore, with the integration of the technologies caused by IoT applications and CPS, it is no longer adequate to treat safety and security separately. The fact that safety can be compromised by security flaws raised the necessity of combining both disciplines. The convergence of safety and security concerns increases the need for industrial facility design and risk assessment approaches that consider both aspects (Kriaa et al. 2015). Consequently, integrating safety and security is a remaining challenge of great importance.

2.4 Cybersecurity

CPS brought a reality where billions of connected devices are placed in several new locations and applications. Likewise, these interconnected systems can generate a knock-on effect of failures and must be secured on all levels. Thus, ensuring security for CPS is a complex problem. These challenges unleash new concerns about cyberattacks and their impact on privacy and safety. Additionally, because they are intentional, security attacks are hard to detect and prevent. They cannot be evaluated quantitatively and do not follow a probabilistic occurrence model (Heinrich et al. 2019).

The standard IEC 27001 (*ISO/IEC 27001:2013* 2013) defines *information security* as the conservation of Confidentiality, Integrity, and Availability of information. *Confidentiality* ensures the information is available only to authorized stakeholders. *Integrity* ensures the accuracy of the information and its completeness. Finally, *Availability* ensures the information is accessible and usable always it is necessary. The security challenges of CPS are usually related to these three properties, although more essential properties have been covered over the years, such as *Authenticity*, that assures confidence in the validity of a transmission, a message, or a message originator.

There are several types of cyber threats. Message corruption, wireless scrambling, eavesdropping, and man-in-the-middle attacks are common examples. Moreover, they come from different sources, like hardware, firmware, data, system applications, and network-related attacks. Over the years, cyberattacks have been detected in several safety-critical fields, such as power distribution, healthcare, military, and transportation infrastructures. The more critical the system is, the more these threats may cause damage.

CPS has additional properties that open a wide range of opportunities to attackers. For example, its real-time behavior means a minimum time change could be enough to put the system into an unsafe state. In addition, critical information can be obtained by exploring wireless network vulnerabilities. IoT devices are usually embedded, battery-powered, and small. Besides, many are low-cost and cannot afford additional power consumption or the necessary code space for expensive mathematical calculations of cryptography methodologies (J. Zhang et al. 2017). Thus, there is an urgent need for a modular security solution to be implemented in CPS to secure the information exchange between them.

This thesis focuses on wireless network threats and proposes defenses for an open transmission system. CPS and IoT devices have bi-directional communication, leaving the link between devices open for network attacks and protocol failure. To ensure and validate the authenticity and integrity of a message, WSSL will implement a mathematical technique known as *Digital Signature*. "Digital signature mechanisms are reasonable in safety-related applications where the credibility of information incoming from (or outgoing to) remote nodes has to be verified" (Rastocny et al. 2016), which is the case of most critical systems. Thus, this approach would allow WSSL to increase cybersecurity for CPS.

2.5 Communication Protocols for CPS

Machine-to-Machine (M2M) communications are becoming crucial for enabling inter-machine communication. "Machine-to-machine (M2M) communications enables networked devices to exchange information among each other as well as with business application servers and therefore creates what is known as the Internet of Things (IoT)." (Aijaz and Aghvami 2015)

Communication protocols are necessary to exchange messages between computing systems. They are responsible for defining a system of rules and digital message formats, allowing different devices to share messages consistently. Dedicated communication protocols are being developed and proposed as middleware to allow M2M communication in the heterogeneity of CPS and IoT systems. They are Message Queue Telemetry Transport (MQTT), HyperText Transfer Protocol (HTTP), Constrained Application Protocol (CoAP), Data Distribution Service (DDS), Advanced Message Queuing Protocol (AMQP), and others.

M2M protocols are well-defined architectures and can be divided between client-server protocol (e.g., CoAP and HTTP) and publish-subscribe protocol (e.g., MQTT and AMQP). The first is also known as the request-response model, commonly used in distributed systems. Here, information is only shared if it is requested. The second is based on forwarding updates from senders (publishers) to interested users (subscribers), which allows information to be shared even if not expressly requested.

2.5.1 Message Queue Telemetry Transport

Message Queue Telemetry Transport (MQTT) is an OASIS standard messaging protocol for the IoT (MQTT 2022), designed as an extremely lightweight publish-subscribe messaging transport with a small code footprint and oriented for connecting remote devices and minimal network bandwidth.

MQTT today is used in several industries, such as automotive, manufacturing, telecommunications, oil, and gas (Silva et al. 2018). In addition, many MQTT brokers are available for testing or real applications, some being free self-hosted brokers. This work will use the Mosquitto Broker, a lightweight open-source broker written in C (Eclipse 2022).

MQTT also provides extra features that are interesting for CPS. It supports TLC/SSL for encrypting connections between the broker and the devices. Plus, it allows the set of permissions since the broker can block the access of forbidden devices to restricted topics. Moreover, MQTT also allows Quality of Service (QoS), providing more reliability by ensuring message delivery.

There are three levels of QoS available:

- QoS 0: At most once delivery
- QoS 1: At least once delivery
- QoS 2: Exactly once delivery

Using MQTT has many benefits and broads the possibility of future analytic work regarding the WSSL performance. MQTT is a communication protocol recommended by ISO/IEC 20922. It is lightweight and efficient, reducing the necessary resources for the user and the network bandwidth. Furthermore, it allows bidirectional communication between devices and servers and broadcast. Finally and indispensable for CPS, it is more scalable than client-server protocols since the broker operations can be highly parallelized, and the messages can be processed by events (event-driven). Adding WSSL and MQTT will help to provide the highest reliability in the worst network conditions.

Chapter 3

Development

This chapter presents the approach to solving the problem defined in the last chapters. Section 3.1 presents the WSSL definition and design, summarizing its concepts and methodologies and how they intend to increase safety and security for CPS. Section 3.2 presents the WSSL architecture and its responsibility in each communication end node, going deeper into the algorithm structure of the library. Finally, Section 3.3 is responsible for breaking through the details of each layer (safety and security) and their respective behavior depending on the side of the communication.

3.1 WSSL Definition

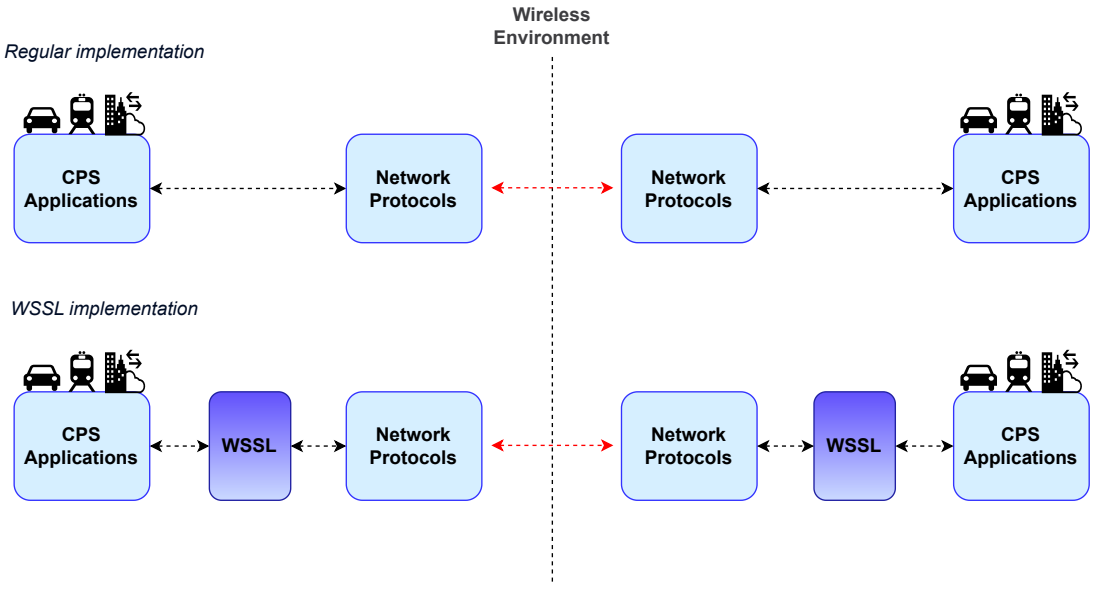


Figure 3.1: Basic implementation of WSSL.

The WSSL consists of an external layer to be implemented in a general wireless communication environment. It implements a detection process for relevant communication issues, establishing a safe and secure connection between each WSSL end-point and providing an extra level of confidence to the CPS devices. Furthermore, the WSSL implementation increases the trust between the Sender and Receiver. So, it avoids that communication failures or malicious interactions could cause critical consequences for the involved agents. It focuses

on open communication systems where the transmission is unsafe. The basic implementation is agnostic, being available for generic use cases independently of the communication protocol, as illustrated in Fig. 3.1.

Thus, WSSL has as its primary function the detection of communications issues between CPS devices, whether or not caused by malicious agents. In this way, WSSL detects communication threats and informs the application, delegating to it the responsibility to act on the problem when necessary. These issues may be message repetition, packet loss, or inter-message delay. By definition, this is assumed to be the safety layer of WSSL. In addition, WSSL implements a message signing model, ensuring increased communication security by allowing the receiver to confirm that the received message comes from the correct Sender (Shallal and Bokhari 2016). This signature model will guarantee the integrity of the received messages, discarding those with data loss. In the security layer, WSSL takes on a detect and discard role for messages with signature problems.

The diagram in Fig. 3.2 exemplifies the network threats and defenses that will be implemented by the WSSL.

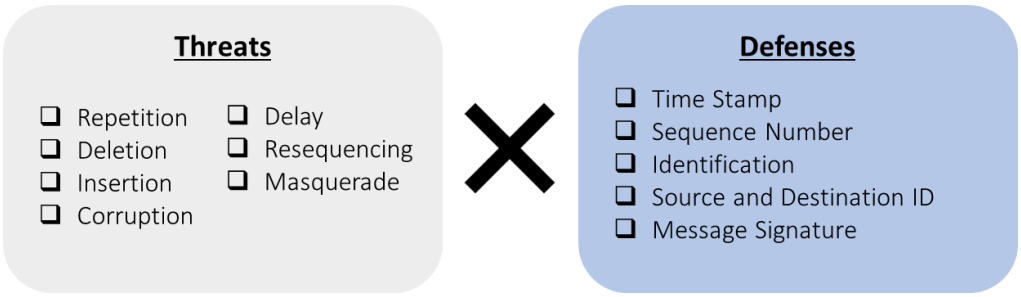


Figure 3.2: WSSL covered threats and defense methods.

Considering the variety of devices and applications that involve CPS, the WSSL development was based on ease of implementation and minimal interference on end devices. Thus, it was decided to build it as a library compiled in C++, allowing it to be attached to most systems on specific hardware or as part of the original system. So, its implementation cost is minimal, and its benefits significant, allowing its application in low-cost or high-performance critical systems.

Furthermore, although it aims to detect the conditions and problems of the communications network, WSSL does not alter the operation of the device where it is implemented, informing the application about the detection of the event so that the device can handle it. In this model, the WSSL works between two end nodes, defined as Sender Device (SD) and Receiver Device (RD). Both SD and RD should instantiate the WSSL library, defining a $WSSL\_Sender$ and a $WSSL\_Receiver$. So, the RD receives data from a finite number of SDs and can also be an SD to other devices.

WSSL library implementation codes are open-source and are available on GitHub repository (Márcia and Enio 2022).

3.2 WSSL Architecture

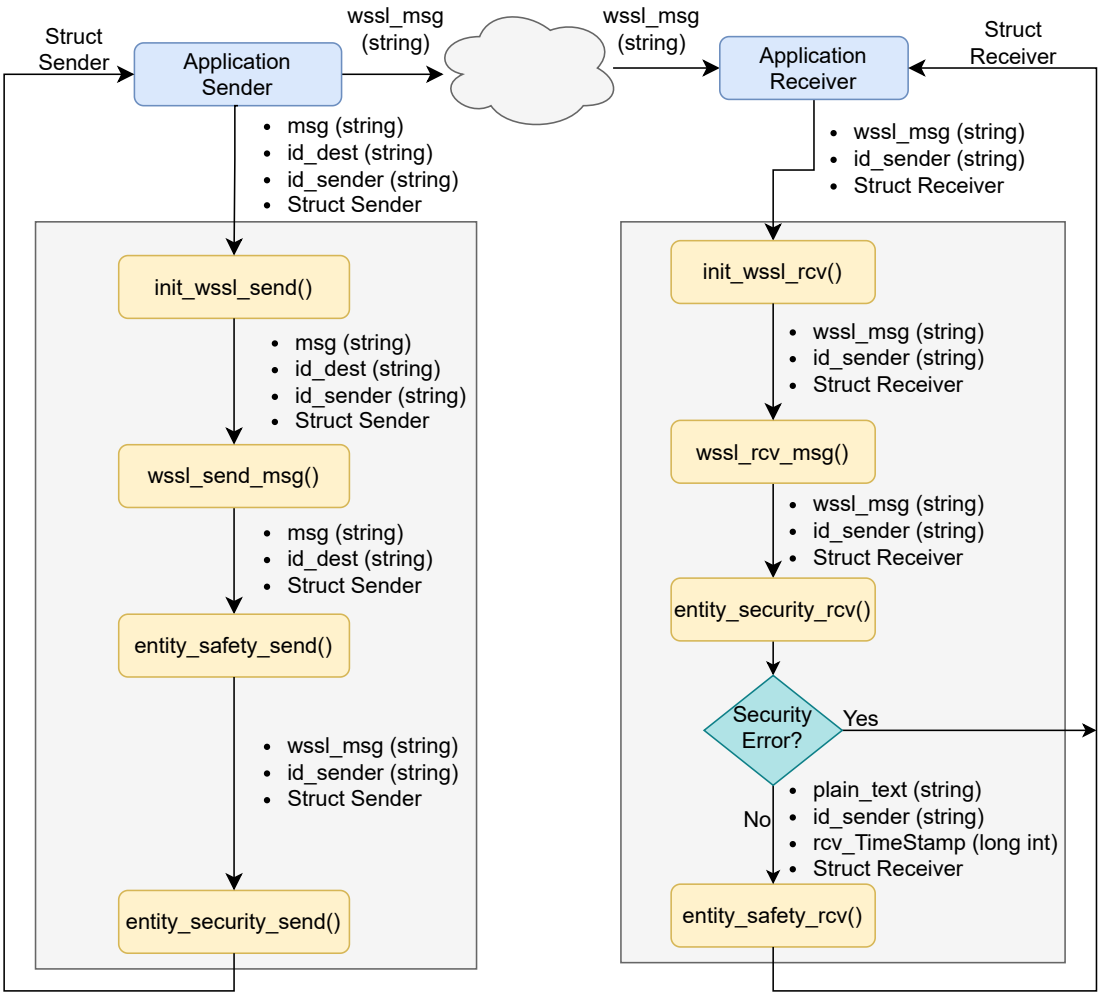


Figure 3.3: General flow of the WSSL Library.

Safety and security mechanisms are implemented as separate layers in each communication end node, and each differs depending on the side (SD or RD). The SD containing the *WSSL_Sender* uses the message content (*msg*), the *RD_ID*, and the *SD_ID* to generate the signed message (*wsslMsg*) and send it to the RD. Conversely, the RD containing the *WSSL_Receiver* must receive from the SD the *wsslMsg* and the *SD_ID* to retrieve the data to the application. Both sides must create a memory table object to store the message information in the WSSL.

This memory table object contains information that can be accessed in the application for both SD and RD. For example, each sender Identifiers (IDs) last message, timestamp, and sequence number are stored in the SDs table. Likewise, the last messages of each receiver ID, its timestamp, sequence number, status, and inter-message delay are stored in the RDs table.

After receiving the data from the SD, the *WSSL_Sender* safety layer adds the safety entities (sequence number and timestamp) to the *msg*, defining a *safeMsg*. Then, the *safeMsg* is transmitted to the *WSSL_Sender* security layer that signs it, adding information about the key and the message size, returning a *wsslMsg* to the application in the SD.

In the proposed architecture, the SD is responsible for sending the *wsslMsg* as an ordinary message. This way, WSSL does not interfere with the device's communication protocol.

The RD application receives the *wsslMsg*. It is vital to notice that, with WSSL, it is impossible to modify the data from the *wsslMsg* without being detected, guaranteeing message integrity and increasing the system's security. So, the RD application calls the *WSSL_Receiver* with the *wsslMsg*. The *WSSL_Receiver* security layer verifies the signature key to check the data signature, and if the SD identity is valid, it removes the signature and gets the *safeMsg*. Finally, the *WSSL_Receiver* safety layer gets the sequence number and timestamp from the *safeMsg*. It returns *msg* to the application, together with the safety information, indicating the inter-message delay and the network status. The general flow of the WSSL Library is illustrated in Fig. 3.3.

After retrieving the safety data from the *safeMsg*, the *WSSL_Receiver* analyses the network message issues from the sequence number and timestamp data received from the SD. The sequence number indicates the following message states: VALID, OUT-OF-ORDER, DUPLICATED, or LOST. Finally, the timestamp is used to calculate the inter-message delay. This delay is calculated only when the message is not out-of-order or if it is not in duplicate. If the delay value exceeds a threshold, the *WSSL_Receiver* will warn the application.

Figure 3.4 illustrates the WSSL messages on the Sender and Receiver. In this example, three equal messages were sent, generating a DUPLICATED status, highlighted in orange. In yellow, we print the safety message generated by the safety layer on the Sender side. In blue, we show the appended information containing the signed message size and the *SD_ID*. Finally, it is possible to observe the signature appended to the safety message in red.

Figure 3.4: Illustration of sending three equal messages.

3.2.1 Signature Method

The signature is developed based on a **public-key signature** using the CryptoIdentity library, designed by Vortex CoLab using Sodium (Libsodium 2022) library as a base. VORTEX CoLab (Vortex-CoLab 2022) is a national entity with the title recognition of Collaborative Laboratory, awarded by the Foundation for Science and Technology (FCT, IP).

In a public-key signature (Mohammed and Varol 2019), a signer generates a key pair consisting of the following:

- A secret key: used to append a signature to any number of messages.

- A public key: used to verify that the signature appended to the message was provided by the creator of the public key.

Notice that all the receivers, or verifiers, need to know and trust entirely a public key forehand so they can verify messages signed using this key. Thus, the keys exchange must be made in a trustable environment. Furthermore, it is essential to underline that adding a signature does not change the representation of the message itself, so this method does not ensure confidentiality.

Sodium library offers two modes of public-key signatures, the *combined mode* and the *detached mode*. The first prepends the signature to the message and saves the signed message in an *unsigned char pointer*. The second saves the signature without attaching a copy of the original message. In this work, we use the combined mode.

Furthermore, the public key (pk) and secret key (sk) are part of the *RD_ID* and *SD_ID*, which are created and stored inside folders during the process of exchanging keys. These folders' path and label are defined by the user when instantiating the library in the SD/RD (see section 3.3.1 and 3.3.2); otherwise, in case they are not defined forehand, are considered the default. Listing 3.1 shows how the identities are created.

```
1 void create_my_id (std::string path, std::string
      label)
2 {
3     cryptoID = new CryptoIdentity(path, label);
4     cryptoID->save();
5 }
```

Listing 3.1: Fuction responsible to create the ID file.

To add a new public key (pk) as a "known identity," it is necessary to call the function **add_known_identity**, passing the path, the pk, and its creator label as parameters. The function is illustrated in Listing 3.2.

```
1 void add_known_identity (std::string path, std::
      string label, const unsigned char *pk)
2 {
3     cryptoID = CryptoIdentity::load(path);
4     cryptoID->addKnownIdentity(label, pk);
5     cryptoID->save();
6 }
```

Listing 3.2: Fuction responsible to a new identity in the known identity list of the ID.

The folders are related to the security identities and contain the *knownIdentites* folder, which stores known identities (public keys) of the RD/SD, a *pk*, an *sk*, an *id*, and a *label*. The path represents the folder's path, and the label is a name to facilitate the identification of the identity, which in this work is used as a synonym of the *senderID* and *destID*. Notice that *id* is a numeric representation of the identity and is not being used in this work.

The folders are illustrated in Figure 3.5. In this example, the paths are *senderFile* and *rcvFile*, and the labels are *SND1* for the SD and *RCV1* for the RD.

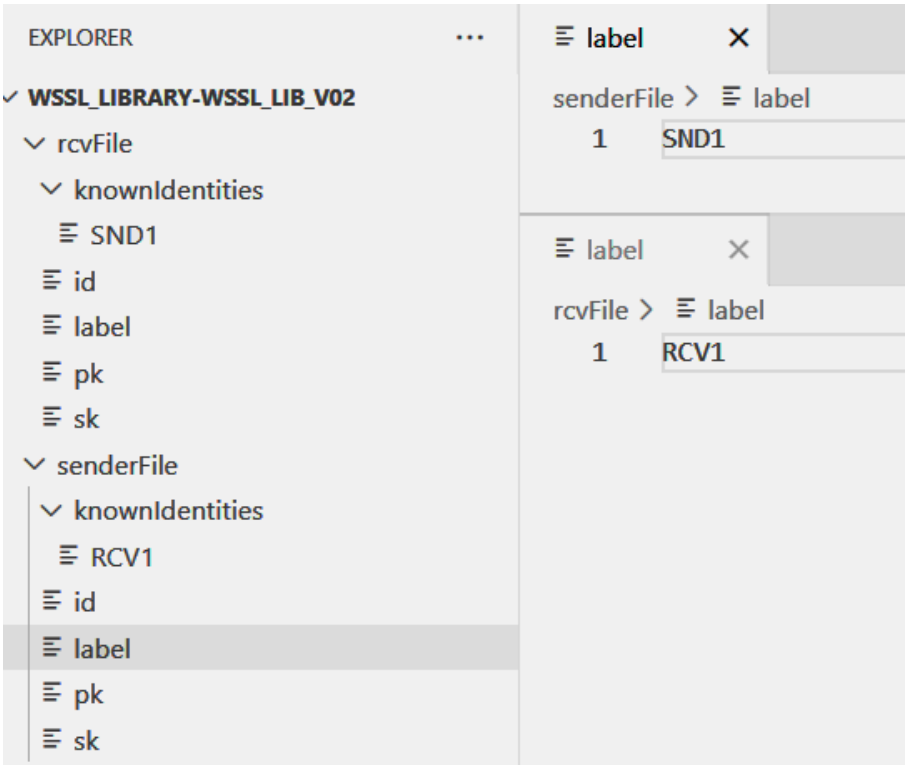


Figure 3.5: Sender and Receiver files containing the Identities information.

The signature works as a 'hash,' so it does not change the representation of the message itself. It has a 64-byte fixed size, verified to ensure the integrity of the message being exchanged in the SD or RD. Also, because it is based on the creator's private key, it provides authenticity, meaning the creator is who it says it is.

In the WSSL library, the SD is responsible for using the method **sign**, which appends the signature to the message. This method throws an error in case the sign method fails. Listing 3.3 shows how this method is used inside the SD Security entity:

```
1 int strSize = safeMsg.length();
2 strSigned = idLoaded->sign(safeMsg, strSize);
3
4 if(strSigned == "error")
5     throw std::runtime_error("Failed to sign
    message!");
```

Listing 3.3: Signing the message coming from safety.

Conversely, the RD is responsible for using the method **verifySignature**, which verifies if the signature appended to the message was issued by the creator of the public key. This method returns an error in case there are any differences in the message received in the RD when compared with the message sent by the SD, representing integrity. It also returns an error if the public keys were not exchanged properly or the identity of the SD/RD is invalid, representing authenticity.

Listing 3.4 shows how the method is used inside the RD Security entity:

```
1 rRet.plainText = idLoaded->verifySignature(
      wsslMsg, idLoaded->getKnownKey(senderID));
```

Listing 3.4: Verifying the signature and recovering the message coming from the SD.

3.3 WSSL Components

As mentioned in section 3.2, the WSSL is an algorithm implemented in a layered structure, which means safety and security mechanisms are implemented as separate layers in each communication end node: the safety entity and the security entity. Each layer behaves differently depending on the component (*WSSL_Sender* or *WSSL_Receiver*).

Furthermore, for optimization of the algorithm and insurance of access control to some critical functions, WSSL implementation was based on classes from which the layers heritage the main characteristics, located in the header file *libwssl_class.h*.

The application has access to two main elements: a table that stores all the data related to the message, defined as a pointer to arrays, and the message itself, the signed message in the RD, and the raw message in the SD. These two elements are inside the class *SenderReturn*, if *WSSL_Sender*, and *ReceiverReturn* if *WSSL_Receiver*, illustrated in Listing 3.5.

```
1  class SenderReturn
2  {
3  public:
4      struct PacketSnd *table;
5      std::string wsslMsg;
6  };
7
8  class ReceiverReturn
9  {
10 public:
11     struct PacketRcv *table;
12     std::string plainText;
13 };
```

Listing 3.5: Return classes of the Sender and Receiver.

Each component has its packet format, *struct PacketSnd*, in case of the *WSSL_Sender*, or *struct PacketRcv*, in case of the *WSSL_Receiver*. These structures represent the information stored inside the object tables, such as data, timestamp, sequence number, *SD_ID* or *RD_ID*, inter-message delays, and message status, and retain the information about the last message sent to an RD or received from an SD, as presented in Listing 3.6.

```
struct PacketSnd
{
        std::string data;
        long int timeStamp;
        int seq_number;
        std::string dest;
        int size_array;
};

struct PacketRcv
{
        std::string data;
        int status;
        long int delay_send;
        long int timeStamp;
        int seq_number;
        int size_array;
        std::string sender;
};
```

Listing 3.6: Packet format of the Sender and Receiver tables.

The WSSL entities for safety and security are based on the class *LibWssl*. This class is responsible for the instantiation of general parameters and the main functions of the WSSL, mainly the ones in charge of initializing the library, the functions related to the security identity and exchange of keys, and the delete functions.

The *init_wssl_snd()*, in case of the SD, and *init_wssl_rcv()*, in case of the RD, are functions called in the application to initialize the WSSL library, and are shown in the Listings 3.7 and 3.8.

```
SenderReturn init_wssl_snd(std::string msg,
      std::string senderID, std::string destID,
            std::string path, SenderReturn sRet);
```

Listing 3.7: Init function for the WSSL_Sender.

```
ReceiverReturn init_wssl_rcv(std::string wsslMsg,
        std::string path, ReceiverReturn rRet);
```

Listing 3.8: Init function for the WSSL_Receiver.

In the case of the *WSSL_Sender*, the *init* function needs to receive as arguments the **senderID**, standing for the Sender label or *SD_ID*, the **destID**, standing for the Receiver label or *RD_ID*, the Sender **path**, and the object of type SenderReturn, whose name is standardized as **sRet**. In the case of *WSSL_Receiver*, the *init* function needs to receive as arguments the **wsslMsg**, standing for the signed message, the Receiver **path**, and the object of type ReceiverReturn, whose name is standardized as **rRet**.

The delete functions, see Listings 3.9 and 3.10, work as watchdogs and are responsible for deleting connections older than 2 seconds from inside the table. The time limit of two seconds is a fixed parameter called **LIM_TIME_SEC**, and since it is considered a critical

parameter, it is not available for the application. Likewise, both delete functions are declared private and cannot be easily modified.

```
1 private:
2 void delete_old_connexions_snd(SenderReturn);
```

Listing 3.9: Functions responsible for deleting old connexions in the table.

```
1 private:
2 void delete_old_connexions_rcv(ReceiverReturn);
```

Listing 3.10: Functions responsible for deleting old connexions in the table.

The variable **sizeLabel** defines the total size of the label for the *SD_ID* and *RD_ID* and is initialized as members of the class with four as default, which means the senderID or destID names have a four-letter pattern (e.g., ID01). The label's size can be changed by the application whenever necessary. The maximum size of the messages (*msgSize*) is fixed in three digits, which means they can vary between 0 - 999 bytes.

Another essential variable is the **delimiter**. WSSL works with string manipulation, which includes splitting and concatenating strings. For that, a delimiter was defined and represented by the standard as two straight slashes ("||").

The declaration of both sizeLabel and delimiter variables is illustrated in Listing 3.11.

```
1 public:
2 int sizeLabel = 4;
3 std::string delimiter = "||";
```

Listing 3.11: Public variables inside the WSSL class.

This delimiter is flexible and must be changed by the user in case it is used inside the messages intending to be sent with WSSL. In the scenario where the user sends the same characters inside the message, WSSL will generate a warning recommending the modification of the variable, and the message will be discarded. This safety procedure, illustrated in Listing 3.12, is necessary to avoid memory corruption within the code.

```
1 try{
2     seqNumber = std::stol(temp1.substr(pos2+Rcv.
    delimiter.length()));
3 }
4 catch (std::invalid_argument const& ex)
5 {
6     std::cout << "[Warning] Invalid message syntax
    . You must not send the delimiter inside your
    message." << std::endl;
7     return rRet;
8 }
```

Listing 3.12: Treating the invalid argument error.

3.3.1 WSSL Sender

The SD only is considered a *WSSL_Sender* when it instantiates the class **WsslSender**. This instance can be created, copied, and overloaded, as illustrated in the five cases in Listing 3.13.

The first way calls the initialization by **default constructor**, responsible for defining a default path, *defaultSender*, and a default label, *defaultLabel*. The second and the third ways call the initialization by **copy constructor**, allowing an instance of the class to be copied to the new instance. The fourth case calls the initialization by the **parameterized constructor** and enables the application to pass the path and label names as parameters. Finally, the last way uses the **copy assignment operator**, allowing the overload of the class instance by another.

```
1 WsslSender Sender;
2 WsslSender Sender2(Sender);
3 WsslSender Sender3 = Sender2;
4 WsslSender Sender4("senderFile", "SND1");
5 Sender = Sender4;
```

Listing 3.13: Different ways to instantiate the WSSL Sender.

After successfully instantiating the class and creating the Sender identity, the SD application calls the *WSSL_Sender* through the function *init_wssl_snd*. First, the *msg* passes through the safety function, *entity_safety_send*, where the safety features are added. Then, the *safeMsg*, the *RD_ID*, the path, and the *SenderReturn* object are passed to the security function, *entity_security_send*. Finally, the *safeMsg* is signed and stored inside the *SenderReturn* object to be returned to the *WSSL_Sender* application. The sequence diagram presented in Fig. 3.8 illustrates the functionality of the *WSSL_Sender*.

WSSL Sender entity for safety

The safety entity, *entity_safety_send*, adds a timestamp and a sequence number to the *msg*, creating and updating the *WSSL_Sender*'s table. In addition, it returns the *SenderReturn* object to be used in the security entity. The timestamp in microseconds is obtained using C++ Chrono library (Cppreferences 2022), a flexible collection of types that track time with different degrees of precision. Figure 3.6 illustrates the general safety flow of the *WSSL_Sender*.

The *SenderReturn* object indexes the sent data with the *RD_ID*. In this architecture, the last sent message ID is compared with the last received one. This strategy creates a **virtual connection** between the SD and the RD while the messages are exchanged. The watchdog constantly checks old connections and drops them after a threshold. Thus, when a message for an RD is received in the *WSSL_Sender*, the Safety entity checks the *WSSL_Sender*'s table, and if there is no connection (previous messages), it creates a new position.

Otherwise, if there is a live connection with this RD, the Safety entity will retrieve the timestamp and sequence number from the previous message. It will update the message data, increment the sequence number, and store this information in the *SenderReturn* object, waiting for the following message. Finally, the *SenderReturn* containing the *safeMsg* is returned for use in the *WSSL_Sender*'s security entity.

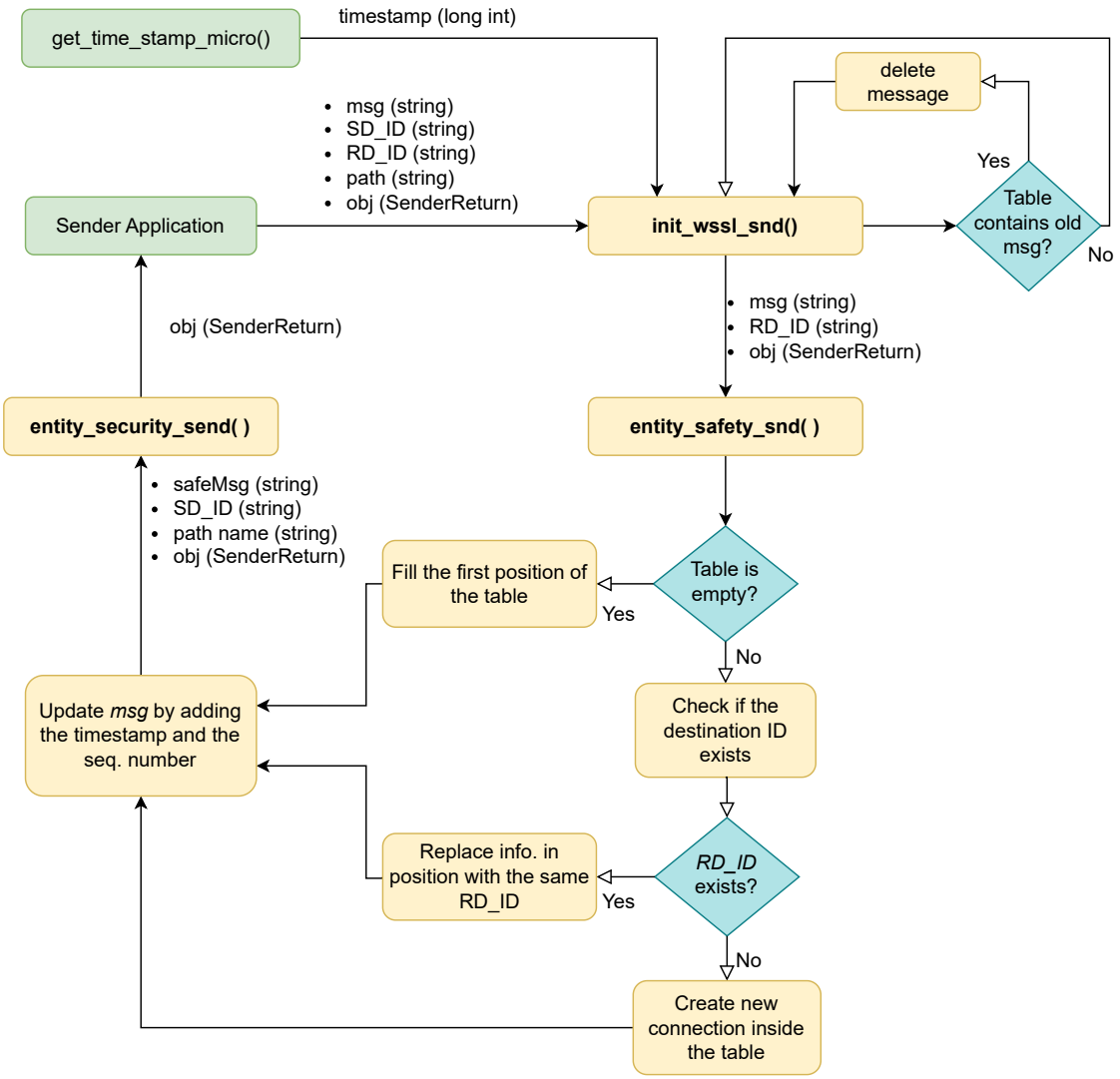


Figure 3.6: WSSL Sender's safety.

The *safeMsg* is a concatenated string with the following format:

Time Stamp || Message data || Sequence Number

WSSL Sender entity for security

WSSL_Sender's security entity, *entity_security_send*, adds a signature to the message received from the safety entity. Each side of the communication has its own public/private key pair. These keys are used to sign and remove the message signatures, ensuring message integrity and authenticity. In this work, we assume the public keys were safely exchanged between the identities using a safe connection, using the protocol described in (Li et al. 2020). Figure 3.7 illustrates the general security flow of the *WSSL_Sender*.

In this work, the method **sign** creates the signature based on the receiver's public key and appends the signature to the *safeMsg*, creating the *wsslMsg*. This method works with messages of any size since it receives a pointer object and size (*strSize*). This strategy increases the system's flexibility due to the non-necessity of padding (Indira, Rukmanidevi, and Kalpana 2020).

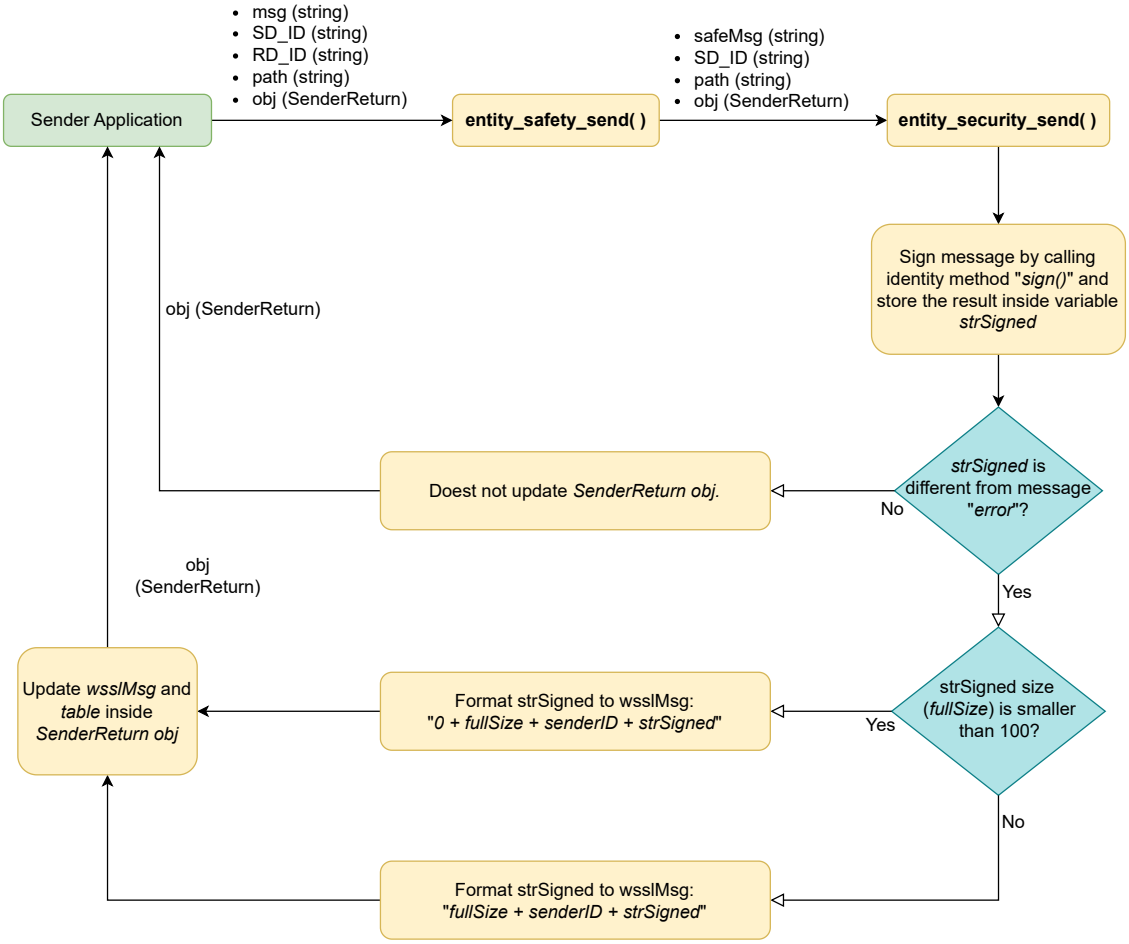


Figure 3.7: WSSL Sender's security.

Moreover, as the *WSSL_Receiver* will need the message size, the Security Entity adds to the *wsslMsg* the *wsslMsg* size (*fullSize*) and the *SD_ID*. Finally, the message appended with this information is stored in the s*SenderReturn* object, which is returned to the application.

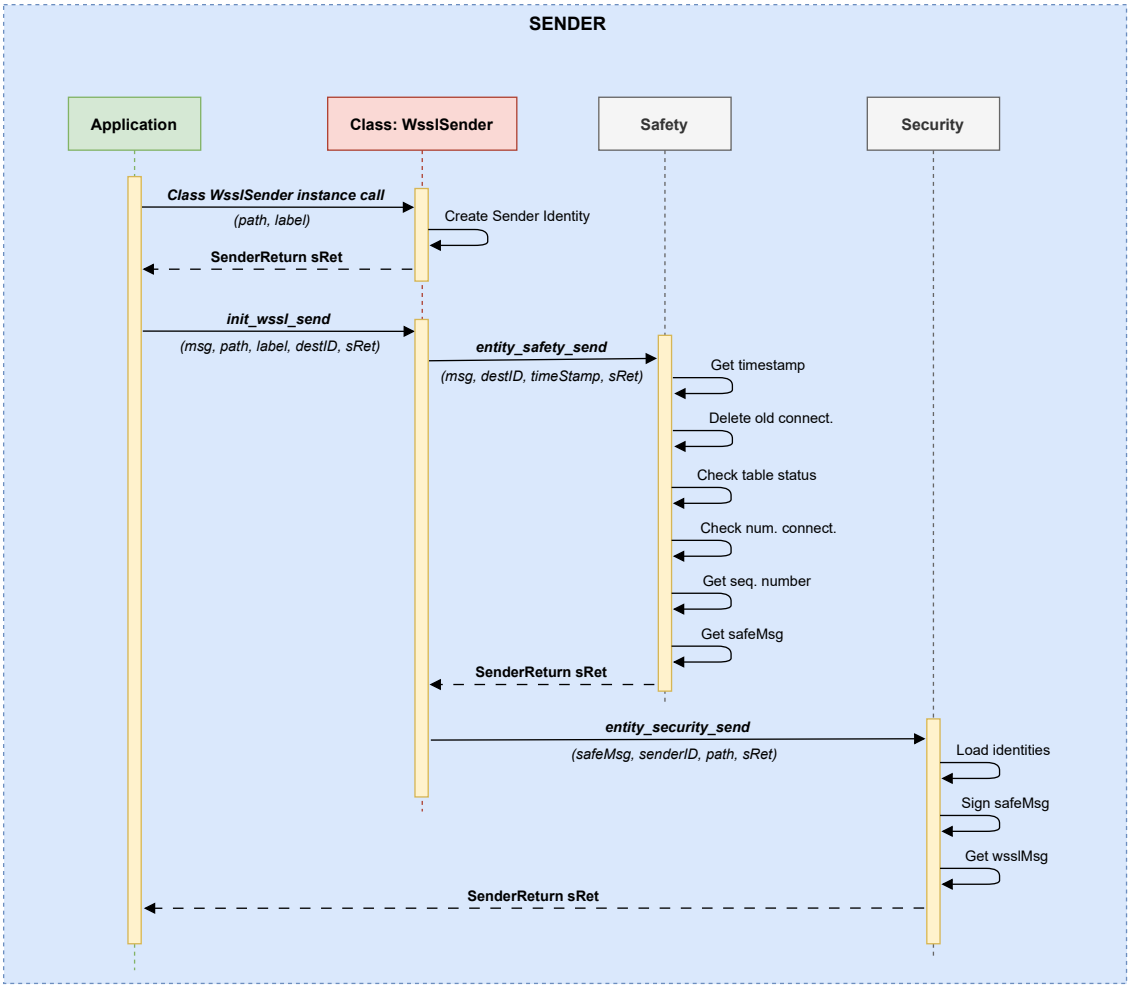


Figure 3.8: WSSL Sender sequence diagram.

3.3.2 WSSL Receiver

The RD only is considered a *WSSL_Receiver* when it instantiates the class **WsslReceiver**. Equivalently to the *WSSL_Sender*, this instance can be created, copied, and overloaded, allowing the initialization by default constructor, responsible for defining a default path, *defaultReceiver*, and a default label, *defaultLabel2*. The initialization by copy constructor, parameterized constructor, and copy assignment operator is also available, as illustrated in Listing 3.14.

```
1 WsslReceiver Receiver;
2 WsslReceiver Receiver2(Receiver);
3 WsslReceiver Receiver3 = Receiver2;
4 WsslReceiver Receiver4("rcvFile", "RCV1");
5 Receiver = Receiver4;
```

Listing 3.14: Different ways to instantiate the WSSL Receiver.

The RD application calls the *WSSL_Receiver*, through the function *init_wssl_rcv*. First, the *wsslMsg* passes through the security function, *entity_security_rcv*, where the signature is removed, and if the *SD_ID* and the signature are correct, the *safeMsg* is recovered.

Otherwise, if the security does not succeed in removing the signature, it returns to the application without executing the safety entity.

Assuming the security succeeded, the *safeMsg* is passed to the safety function, *entity_safety_rcv*, where the *msg* is separated from the timestamp and the sequence number and retrieved to the *WSSL_Receiver* application. The sequence diagram (Fig. 3.11) illustrates the *WSSL_Receiver* functionalities.

WSSL Receiver entity for security

WSSL_Receiver's security entity, *entity_security_rcv*, is responsible for removing the signature from the received *wsslMsg*. The signature has a fixed 64-byte size, and *fullSize* is obtained from the data appended by the Sender to the *wsslMsg*. The general flow of the *WSSL_Receiver*'s security entity is shown in Fig. 3.9.

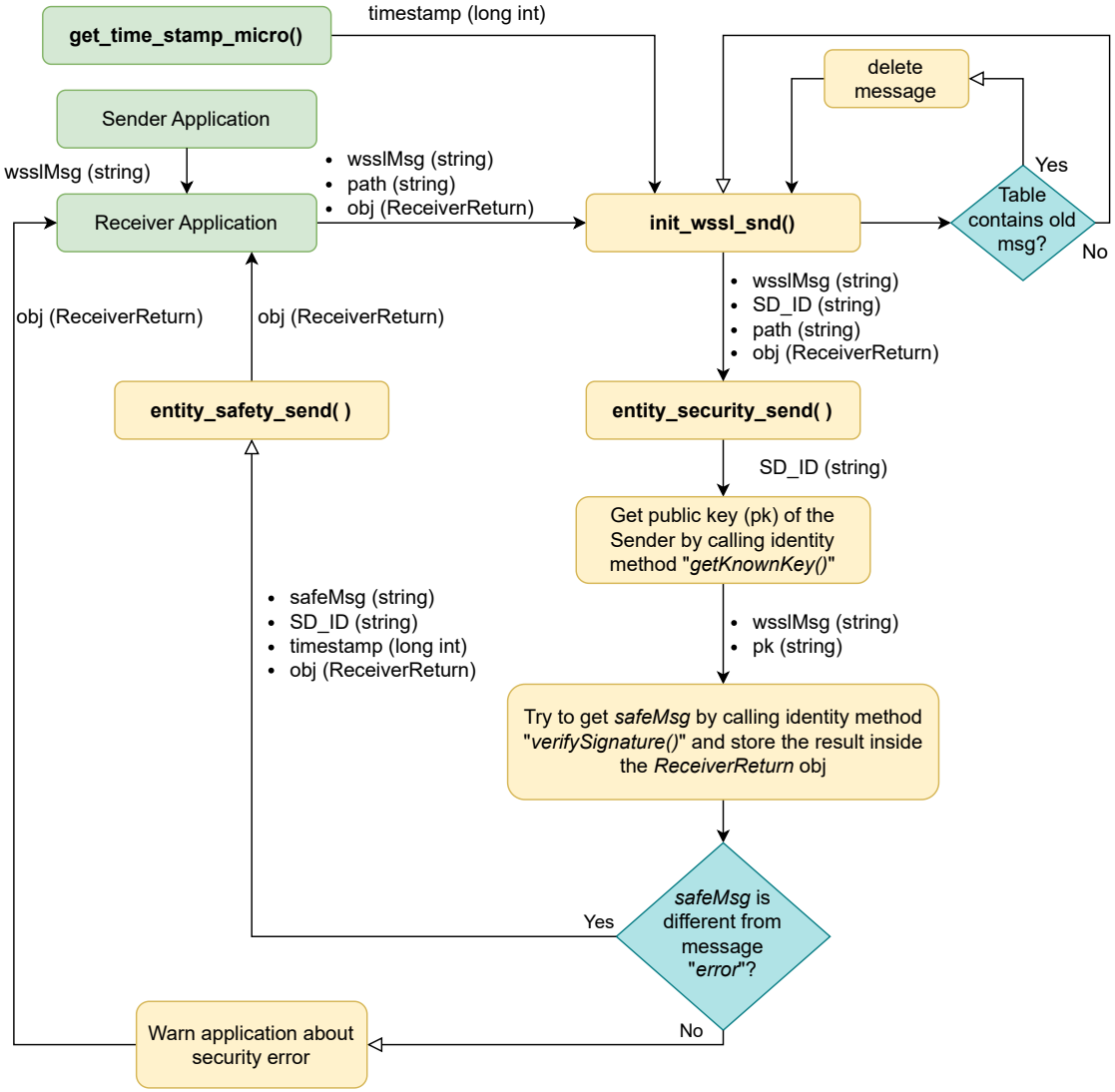


Figure 3.9: WSSL Receiver's security.

Afterward, the method **verifySignature** is used to remove the signature. This method verifies if the *SD_ID* is correct and, if everything is veracious, uses this ID to remove the

signature from the *wsslMsg*. At last, the *safeMsg* is retrieved and returned for use in the *WSSL_Receiver*'s safety entity.

WSSL Receiver entity for safety

WSSL_Receiver's safety entity, *entity_safety_rcv*, has a few different functionalities compared to *WSSL_Sender*'s safety, whereas it is necessary to make some treatment related to the message's integrity. These differences are in the check conditions, where each received message is designated with a status related to the sequence number, and in the inter-message delay. The general flow of the *WSSL_Receiver*'s safety entity is shown in Fig. 3.10.

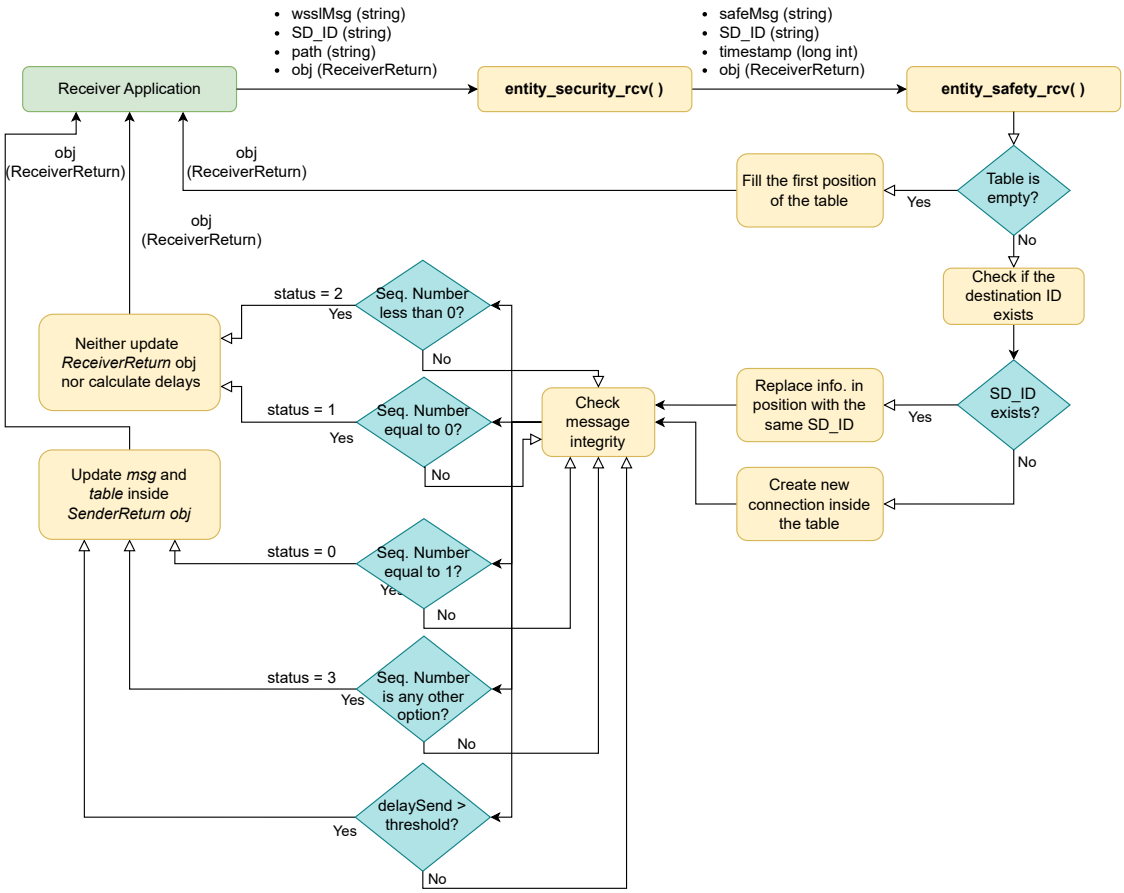


Figure 3.10: WSSL Receiver's safety.

The inter-message delay, **delaySnd**, represents the time delay between the last two received messages. This delay is quantified when the table is not empty, and the *SD_ID* is equal to any ID inside the table; Otherwise, the delay is set to zero. This delay is calculated by subtracting the *msg* timestamp from the virtual connection timestamp with the same *SD_ID*. When the delay value is bigger than a defined threshold, a warning is generated for the application, but the status of the message is not changed. In this way, WSSL leaves the decision to the application to use or not the message.

Similarly to the *WSSL_Sender*'s safety entity, when a message is received in the *WSSL_Receiver*, if there is no connection inside the table, the safety entity creates a new position, filling it with the retrieved data.

Both inter-message delay and sequence number are checked depending on the table status. If the table is empty or it is the first connection with the corresponding *SD_ID*, the only status checked is LOST. Alternatively, if the table has live connections, the sequence number is checked for all the cases introduced in section 3.2. Also, only the last message of each *SD_ID* is stored inside the table.

If the *RD_ID* differs from the existing IDs and there is no DUPLICATED or OUT-OF-ORDER message status, a new connection is created inside the table, and a new delay is calculated. A VALID status occurs if the delay is not bigger than the threshold and the difference between the received and the last sequence numbers equals 1. A DUPLICATED status is set when the difference between the sequence numbers equals 0. An OUT-OF-ORDER status is defined when the difference between the sequence numbers is less than zero (0). Last, LOST status happens when the difference between the sequence numbers is different from the other cases.

Each message status generates a unique table event, which is returned to the *WSSL_Receiver* application. DUPLICATED messages generate status 1, OUT-OF-ORDER messages generate status 2, and LOST messages generate status 3.

The data inside the table is organized in the following format:

Timestamp || Msg. data || Seq. Number || Status || delay

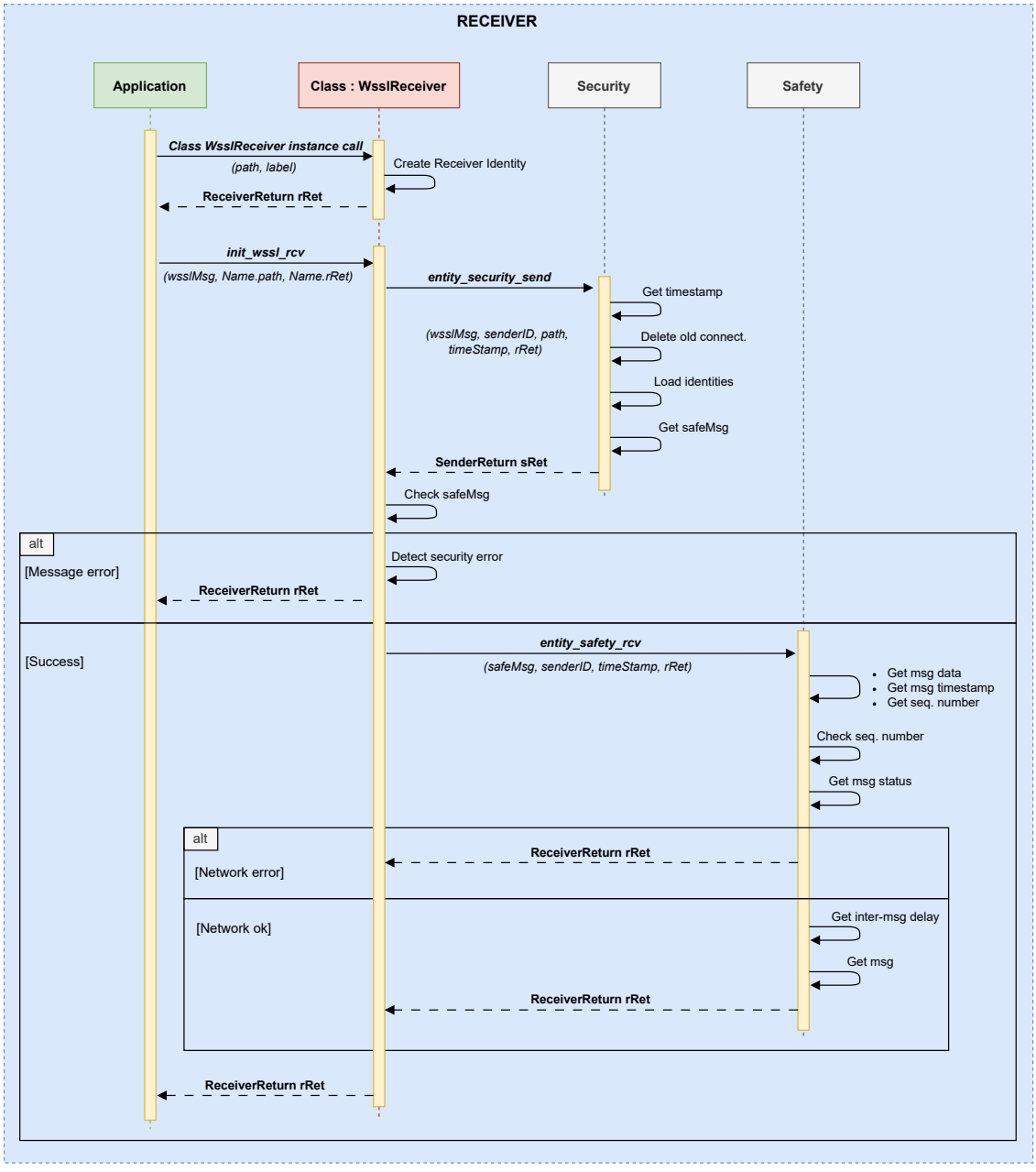


Figure 3.11: Receiver sequence diagram.

Chapter 4

Evaluation

This chapter presents the methodology, analysis, and results of the evaluation tests. Section 4.1 reveals the laboratory setup, techniques, and tools that will be used during the development of these tests. Next, Section 4.2 explains the methods to evaluate WSSL using MQTT protocol, testing the error detection and costs related to the library implementation, presenting the gathered data, and analyzing the results. Finally, Section 4.4 summarizes the results and concludes the chapter.

4.1 Methodology

Since this work is developed for Critical Systems, it is crucial to evaluate WSSL's efficiency and reliability, calculate its implementation costs, and show its capability to detect errors. To assess the WSSL's efficiency, and because $WSSL\_Receiver$ must not fail when treating and verifying the integrity of the message, some errors will be forced into the library.

The evaluation tests will be based on two steps. In this chapter, we cover the first step, that is, integrate WSSL in a more controlled environment by implementing the MQTT protocol to establish two-way communication between two desktops, as illustrated in Fig. 4.1.

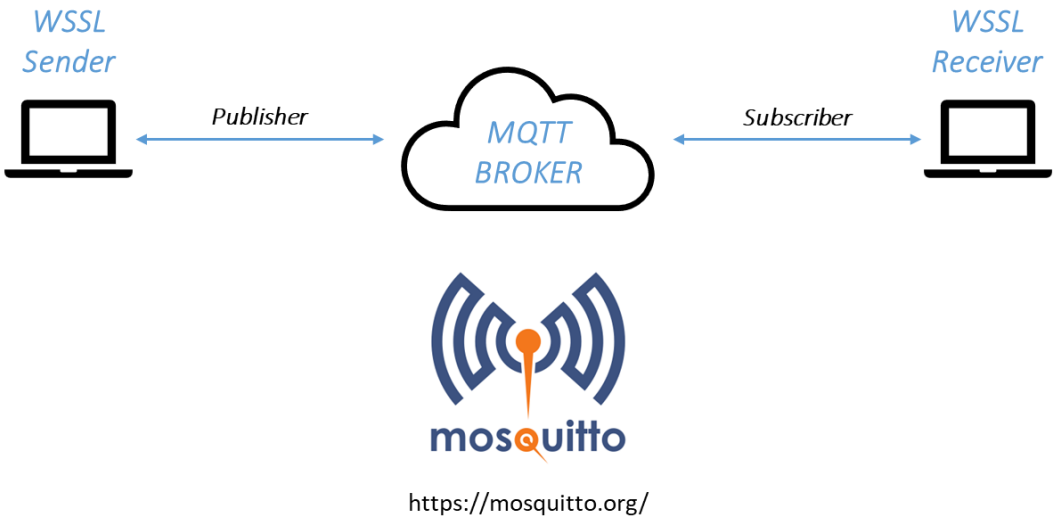


Figure 4.1: Laboratory using MQTT and WSSL

During the second step, presented in Chapter 5, WSSL will be implemented inside the application covering an authority Handover in the ADACORSA project. Both cases will test WSSL implementation costs and its ability to detect network errors and warn the application efficiently: the first in a more controlled environment and the second in a simulation environment.

For this purpose, a laboratory setup will be put together using two desktops running Ubuntu version 20.04. Each desktop will be responsible for a communication end node, where a desktop with an Intel Pentium CPU G645 2.90 GHz processor and 4 GB of memory RAM will contain the *WSSL_Sender* and a desktop with an Intel Core i5-9300H 2.40 Ghz processor and 16 GB of memory RAM will contain the *WSSL_Receiver*.

4.2 Evaluation using MQTT

To evaluate the algorithm in this controlled environment, two metrics and an error injection model for sending and receiving messages were defined. Thus, we initially assessed the cost of adding WSSL in sending and receiving messages, comparing the original system and the same system with WSSL. In this same scenario, we evaluated the time between messages, given the importance of this metric in validating CPS. Finally, we define a set of errors to be inserted into the communication system and show how WSSL can detect and handle these events.

The following tests were made using the MQTT messaging protocol over the Transport Layer (Silva et al. 2018), with the *Mosquitto Broker*, an open-source and lightweight message broker suitable for use on all devices. Using MQTT, the publisher must be configured as the *WSSL_Sender*, as illustrated in Listing 4.1, and the subscriber as the *WSSL_Receiver*, as illustrated in Listing 4.2.

```
1 void publish_messages(WsslSender *Sender, struct mosquitto
      *mosq)
2 {
3     Sender->sRet = Sender->init_wssl_snd(myMessage,
          Sender->label, destID, Sender->path, Sender->sRet
      );
4     ...
5
6     mosquitto_publish(mosq, NULL, "wssl",
          Sender->sRet.wsslMsg.length(), (const void*)
                  Sender->sRet.wsslMsg.c_str(), 0, false
      );
7 }
```

Listing 4.1: Function publishing in topic "wssl" in Mosquitto broker.

```
1 void on_message(struct mosquitto *mosq, void *obj, const
      struct mosquitto_message *msg)
2 {
3     string wsslMsg(static_cast<const char*>(msg->payload),
            msg->payloadlen);
4     ...
5
6      Receiver.rRet = Receiver.init_wssl_rcv(wsslMsg,
            Receiver.path, Receiver.rRet);
7 }
```

Listing 4.2: Function subscribed in topic "wssl" in Mosquitto broker.

To validate error detection, the highest frequency was chosen with a guarantee of no data loss. The goal was deliberately sabotaging the message to check if everything was working as it should. For example, delaying a message and checking whether the WSSL will detect it could prove that WSSL will alert the application if it exceeds the defined delay threshold. Then, using statistical methods, it will be possible to analyze whether the results are admissible for CPS.

Six types of security errors are going to be tested:

- Invalid signature;
- Invalid SD;
- Inter-message delay bigger than the threshold;
- DUPLICATED message;
- OUT-OF-ORDER message;
- LOST message.

WSSL involves time-critical messages and has to fulfill real-time constraints. Besides, it aims to be lightweight and interfere as little as possible in the system. Therefore, the time to send all messages, the time to receive all messages, and the inter-message delay must be evaluated.

The tests for measuring the time to send all messages and to receive all messages were repeated ten times for each of the following cases:

- Using the WSSL entities (Safety and Security) together;
- Using only the safety entity;
- Using only the security entity;
- Using only the application without WSSL;

On the other hand, to evaluate the inter-message delay, some messages were purposely delayed, and the graph of WSSL detection capability was analyzed.

4.2.1 WSSL Cost

WSSL involves time-critical messages and has to fulfill real-time constraints. Besides, it aims to be lightweight and interfere as little as possible in the system. Therefore, WSSL costs and its capability to keep track of the inter-message delay must be evaluated.

WSSL costs are evaluated with tests based on the overhead that WSSL adds to the application and are calculated considering the time to send all messages and the time to receive all messages. An overhead investigation is vital to trace an acceptable cost region to use WSSL in different applications, depending on its criticality and real-time parameters. The relevant data containing the sent and reception time when adding WSSL into the communication between devices are described in a database also found in the GitHub Repository (Márcia and Enio 2022) in the Excel Sheets inside the folder "dataBase".

Table 4.1 is an example of the gathered data using a frequency of 1000 Hz while sending fifty thousand messages. The *sent time* is considered the total time the $WSSL\_Sender$ takes to send all messages, and the *reception time* is the total time the $WSSL\_Receiver$ takes to receive all messages. Both are calculated by subtracting the timestamp of the first message from the last message being sent (or received).

Table 4.1: Sent and reception time costs when sending fifty thousand msgs with the frequency of 1000 Hz.

	MQTT		**Safety**		**Security**		**Safe. and Sec.**	
Tests	**Sent time (ms)**	**Rcpt. time (ms)**	**Sent time (ms)**	**Rcpt. time (ms)**	**Sent time (ms)**	**Rcpt. time (ms)**	**Sent time (ms)**	**Rcpt. time (ms)**
T1	55928	55929	+515	+514	+8239	+8238	+8403	+8405
T2	55940	55941	+485	+486	+8320	+8324	+8472	+8474
T3	55858	55859	+545	+546	+8334	+8338	+8419	+8419
T4	55825	55825	+554	+557	+8372	+8378	+8484	+8490
T5	55885	55886	+523	+523	+8284	+8287	+8470	+8475
T6	55876	55878	+622	+621	+8249	+8256	+8474	+8476
T7	55879	55880	+543	+543	+8197	+8202	+8559	+8563
T8	55883	55884	+477	+477	+8239	+8241	+8376	+8381
T9	55948	55949	+522	+519	+8175	+8179	+8290	+8293
T10	55867	55867	+601	+603	+8260	+8264	+8394	+8396
Avg	**55889**	**55890**	**+539**	**+539**	**+8267**	**+8271**	**+8434**	**+8437**
%	**100,0**	**100,0**	**+1,0**	**+1,0**	**+14,8**	**+14,8**	**+15,1**	**+15,1**

These tests were repeated for different amounts of messages and frequencies to analyze the performance of WSSL in different situations. Therefore, a thousand, ten thousand, fifty thousand, and a hundred thousand messages were sent at four different frequencies: 1000 Hz, 2000 Hz, 5000 Hz, and 10000 Hz. The results are described in Section 4.4, which shows in Fig. 4.5, 4.6, and 4.7, the costs of the safety entity, security entity, and of WSSL, that is, both layers together. Furthermore, the charts in Figure 4.4 summarize the average time costs progression regarding the number of messages in milliseconds of WSSL, taking the MQTT results as a reference. During the tests, there were no lost messages.

4.3 Inter-message delay

WSSL's ability to detect the inter-message delay, which is the delay between the message saved in an existing connection inside the table and the last message received with the same ID, was also evaluated. The purpose is to show that WSSL is aware of the network overhead and will alert the application if the delay exceeds the defined delay threshold.

The delay detection accuracy was tested by sending 100 messages at a frequency of 10000 Hz and a threshold of 10 ms. The messages multiples of eleven were delayed by 11 ms, so it would be possible to simulate an inter-message delay in *WSSL_Receiver*. The results of the delay detection are plotted in Fig. 4.2.

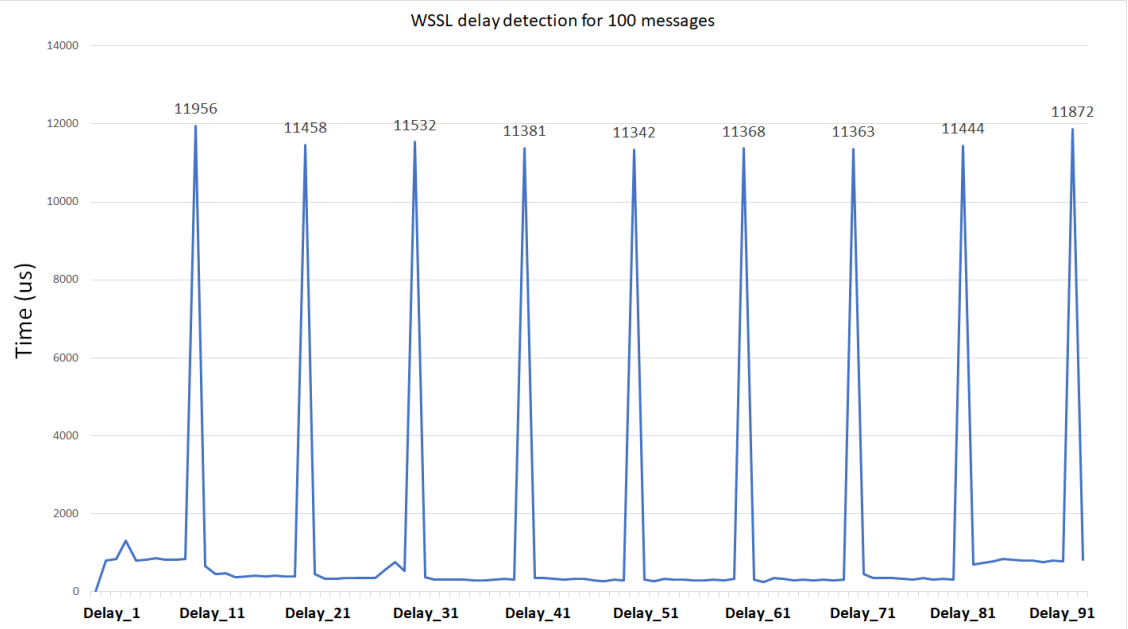


Figure 4.2: WSSL's delay detection - MQTT.

4.3.1 Error detection

For evaluation of the WSSL's efficiency, some errors were forced into the library. Because *WSSL_Receiver* must not fail when treating and verifying the integrity of the message, in this step of the project, the goal was deliberately sabotaging the message to check if everything was working correctly.

During the tests, we sent twenty messages with the same *SD_ID*; within these messages, some samples were chosen to be purposely "sabotaged", as shown in Fig. 4.3. The delay threshold was considered ten milliseconds, and the following six types of security errors were tested:

- **(A)** Invalid signature (*Security error*) at message 2;
- **(B)** Invalid SD_ID (*Security error*) at message 4;
- **(C)** Inter-message delay bigger than the threshold (*status = 4*) at message 6;
- **(D)** DUPLICATED (*status = 1*) at message 8;
- **(E)** OUT-OF-ORDER (*status = 2*) at message 9;

- **(F)** LOST (*status = 3*) at message 15.

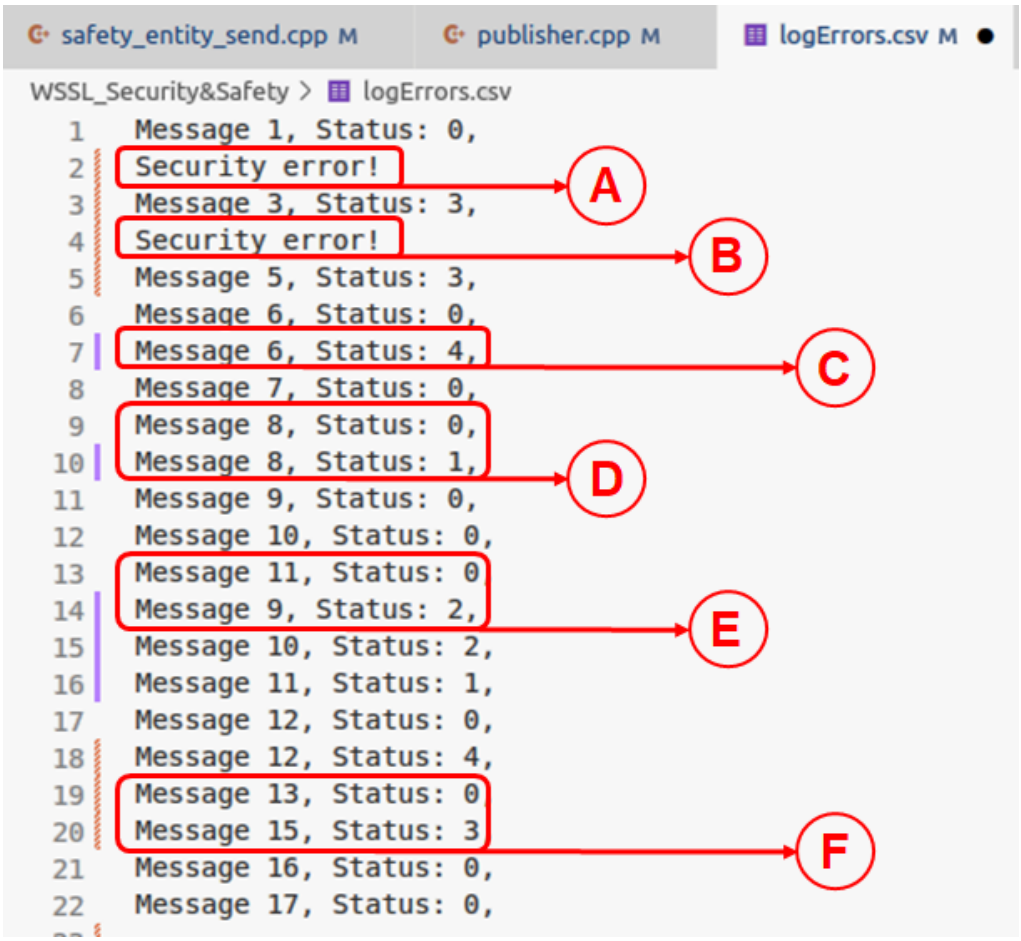


Figure 4.3: Log file of generated errors.

Notice that the statuses are WSSL's method for notifying the application about the message condition, so handling the error must be the application's responsibility. Also, the statuses related to the delay are assigned as four. They are only printed to illustrate the system behavior since the delay is analyzed separately by *WSSL_Receiver* and does not change the message status. When it arises simultaneously with the other warnings, it generates two different prints for the same message, which is the case of messages 6 and 12. Conversely, when the message is sent twice, it ends with DUPLICATED status, which is visible in message 8.

Finally, each error can consequently trigger other errors. For instance, it can be noticed in messages 3 and 5 once Security invalidated messages 2 and 4. Also, message 12 took too long to arrive because the previous messages were out of order and were not being saved into the *WSSL_Receiver* table.

4.4 Tests Results and Conclusions

The evaluation presented in the last section allowed us to asses and gather significant results regarding WSSL. The analysis of Fig. 4.3 demonstrates the ability of WSSL to detect all possible attacks, caused or not by malicious agents, on the receiving device. Furthermore, it is capable of detecting and informing the application about network threats arising from these attacks or even network congestion.

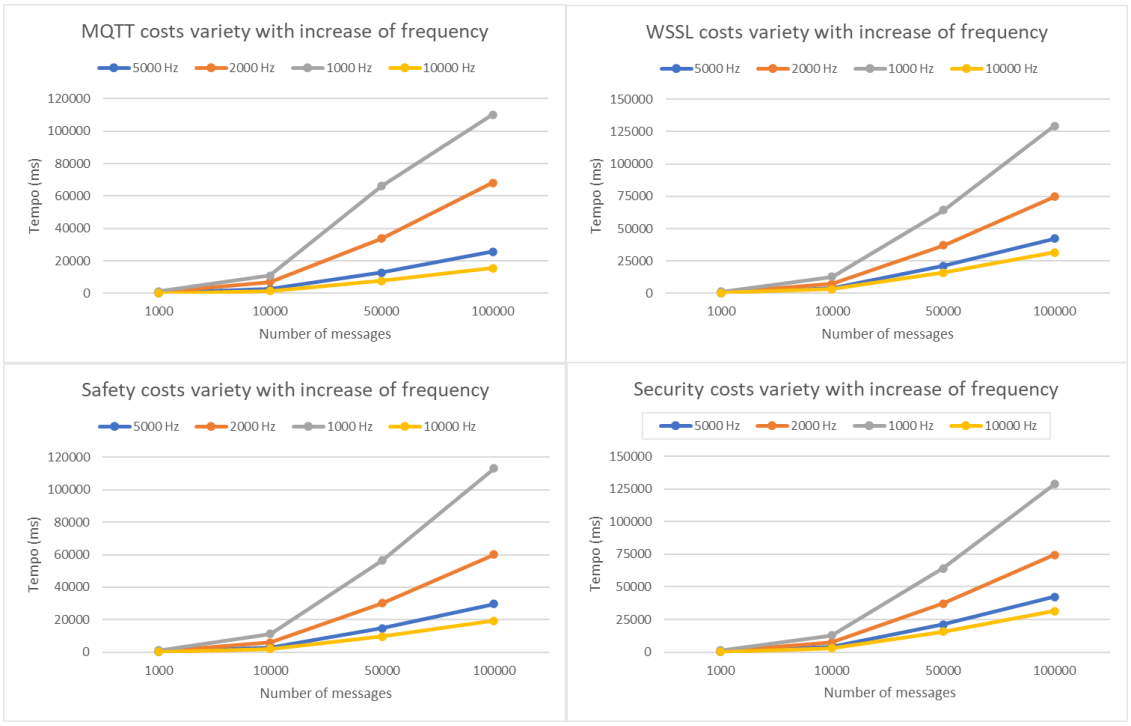


Figure 4.4: WSSL average costs versus frequency and number of messages when integrated with MQTT.

Additionally, when analyzing the data and the graphics 4.5 and 4.6, it is possible to conclude that most of the overhead comes from WSSL's security entity. In contrast, the safety entity increases a little less than 5% of the overhead in the worst-case scenario. This behavior was expected, considering we use a public-key signature, and related cryptography methods usually have a high overhead.

The analysis of the graph 4.7 allowed us to determine that starting with 1000 Hz, WSSL only costs around 15% to 17% of overhead and that the costs tend to decrease to less than 10% when using lower frequencies. This tendency can be noticed even better during the tests made with ADACORSA, which will be described in the next chapter.

Although, even with costs around 17%, the frequencies used during the tests were way higher than the necessary to fulfill hard real-time systems, such as automotive (ETSI TR 102 638 2009) and aerial applications (Yildiz et al. 2021). For example, the European Telecommunications Standards Institute (ETSI) standard TR 102 638, a document that defines a Basic Set of Applications mainly focusing on communications in the Vehicle-to-Everything (V2X) dedicated frequency band, defines several critical parameters for cooperative road safety related to different situations: Cooperative awareness (e.g., Signal violation warning, Overtaking vehicle warning, Lane change assistance), Cooperative collision avoidance

or mitigation (e. g. Across traffic turn collision risk warning), Road hazard warning (e.g., Hazardous location notification), and many others. Regardless of the application, the minimum frequency of the periodic messages is between 1 Hz to 10 Hz, much less than used during tests.

Finally, the graph analysis plotted in Fig. 4.2 shows efficiency and precision in detecting inter-message delay variations, which is important for monitoring the network quality and out-of-pattern behaviors.

Therefore, implementing WSSL over a commonly used protocol for IoT applications such as MQTT reinforces its flexible character for other protocols and possible use in different conditions. In this sense, the results obtained by sending a large packet of sequential messages without a significant increase in communication delay show the viability of its application based on the architecture proposed in this work.

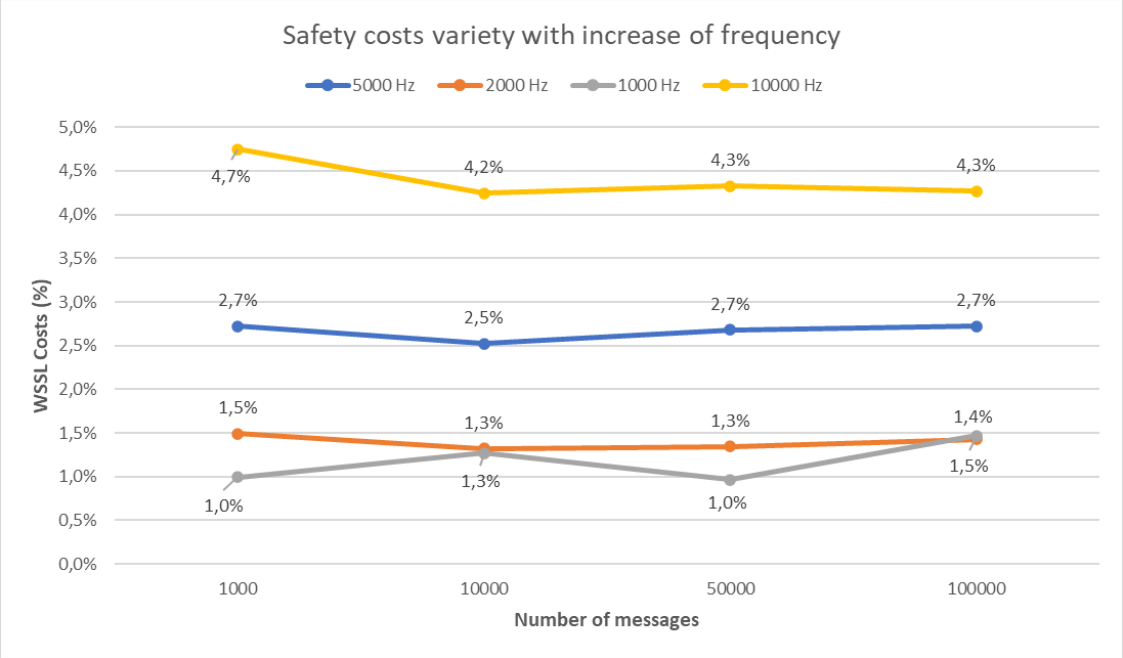


Figure 4.5: WSSL's safety entity costs in percentage versus frequency and number of messages.

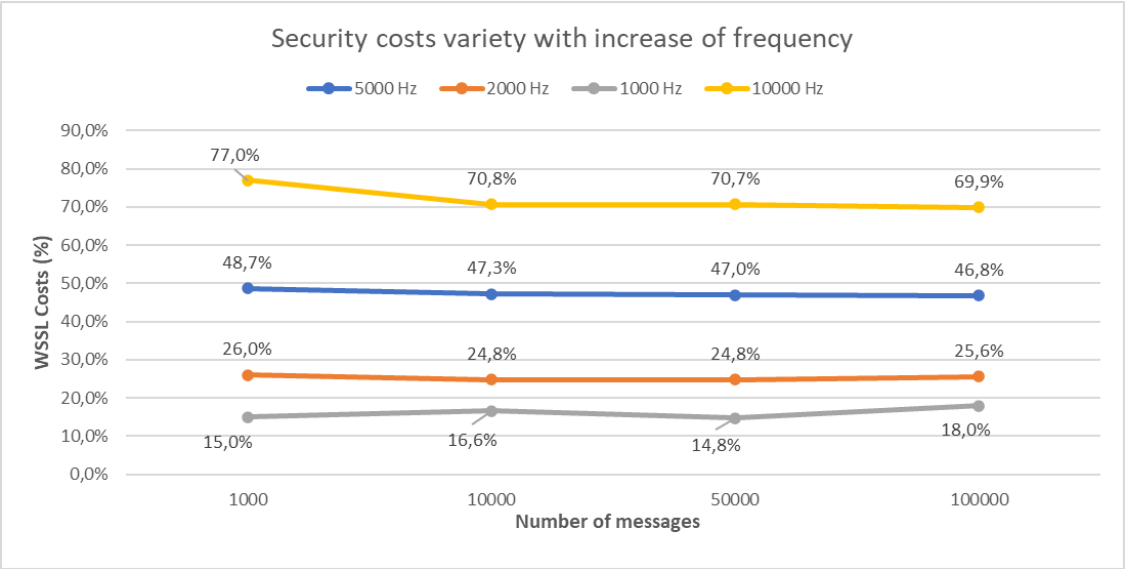


Figure 4.6: WSSL's security entity costs in percentage versus frequency and number of messages.

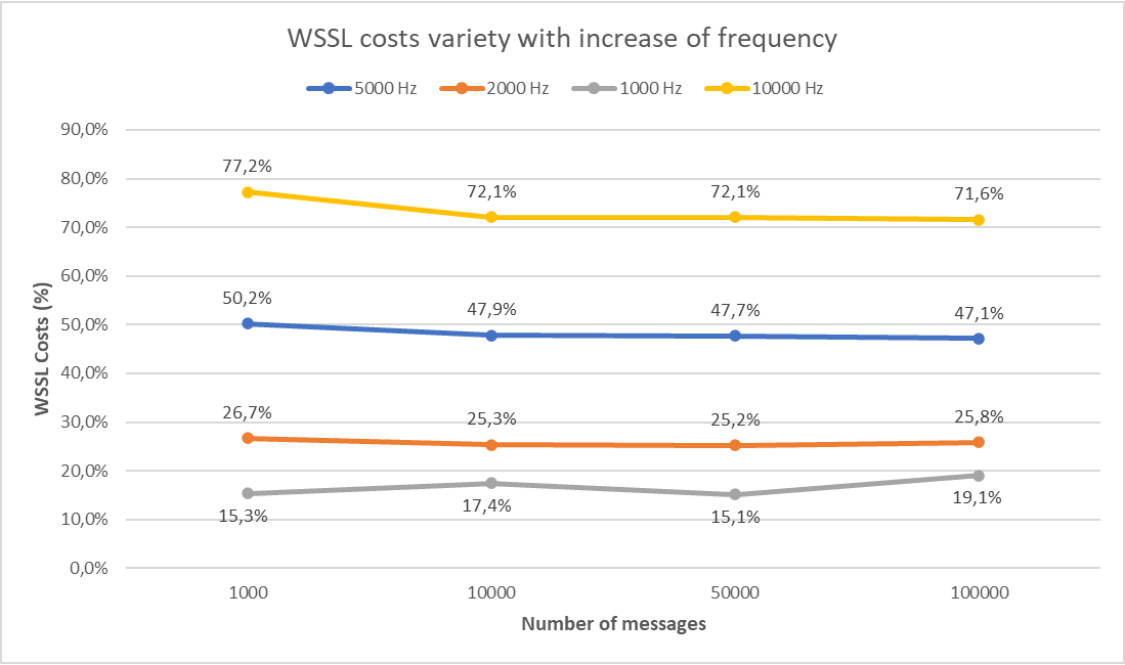


Figure 4.7: WSSL costs in percentage versus frequency and number of messages.

Chapter 5

Applications

As mentioned in the last chapter, WSSL was implemented and integrated with an Authority Handover related to the project ADACORSA. This integration allowed it to be tested in a more unconstrained environment, allowing future implementation from the simulation to an actual application. As a result, new tests regarding WSSL latency had to be implemented to evaluate the impact of integrating both systems. Similarly to the tests using MQTT, it includes qualitative and quantitative methods, measuring WSSL costs, the capability to detect the delay between messages, and others. Section 5.1 will clarify why the initial idea regarding the WSSL integration and evaluation within CopaDrive changed to the current ADACORSA project described later in this chapter. Section 5.2 presents the new idea and describes the implementation within ADACORSA project, going deeply into the tests developed to assess the viability of this integration.

5.1 WSSL and CopaDrive

Initially, WSSL would be integrated with CopaDrive. CopaDrive is a framework for simulating and validating vehicle control and communication strategies, using the Robot Operating System (ROS) as a base. Intending to evaluate vehicle communication capabilities, CopaDrive simulates vehicles in 3D using Gazebo and simulates communications using OMNET++, as illustrated in Fig. 5.1. The communication is modeled on ETSI ITS-G5, based on IEEE 802.11p, adopted in Europe as a standard (ETSI EN 302 663 2012). In this standard, vehicle information is exchanged through messages called Cooperative Awareness Message (CAM), implemented through the Artery project (Riebl 2020).

WSSL would act as an intermediate layer responsible for establishing and guaranteeing vehicle connection, as illustrated in 5.2. Also, a defensive countermeasure if WSSL detects a problem was thought to be implemented into the CopaDrive simulation. However, during the implementation of CopaDrive, it was noticed that some limitations would be reached regarding the time for developing this thesis.

The CAM protocol from ITS-G5, used inside CopaDrive aimed to be integrated with WSSL, demonstrated some characteristics that would represent a challenge when working with WSSL. First, CAM has a specific package format that would be necessary to modify and create new messages to be exchanged to allow integration. The main drawback was that between the data being shared with Omnetpp and ROS, there is a mixing between code in C and C++, demonstrating some limitations on creating a new field inside the platooning CAM data. The intercommunication between several structs makes it difficult to modify the codes in C to C++ so that they would accept the messages type from C++, in this case, the string format.

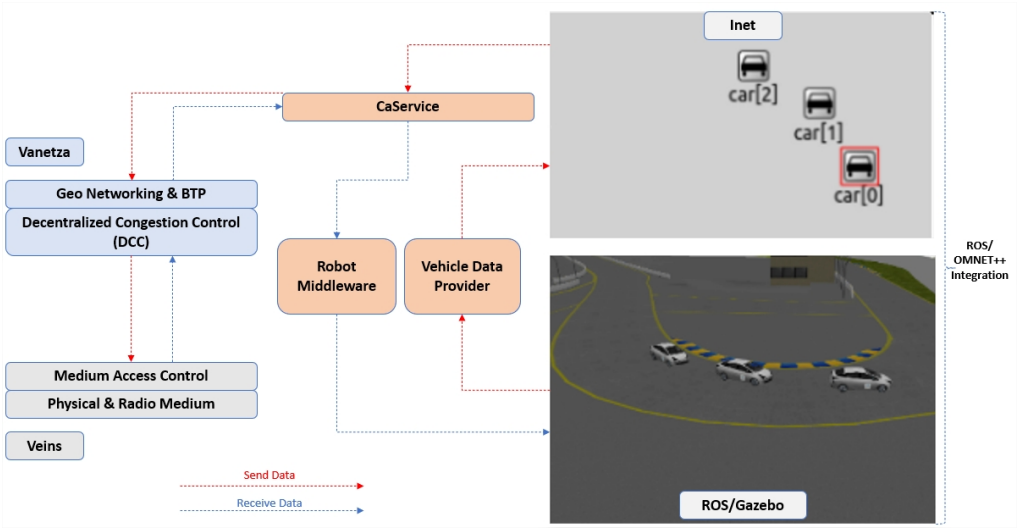


Figure 5.1: CopaDrive Architecture (Filho, Severino, Rodrigues, et al. 2021).

Within the limited time to develop this work, this kind of modification was thought to be unfeasible or to limit the time supposed to be invested in a more extensive evaluation. For instance, there was a possibility of modifying WSSL to work with char arrays. However, when working with a char array in C, we would be limiting WSSL since it would be necessary to allocate memory for the message, requiring the application to define the message size from the beginning or define a specific and fixed size for the array inside the library. This constraint does not make sense since WSSL aims to be generic and cover a variable message size.

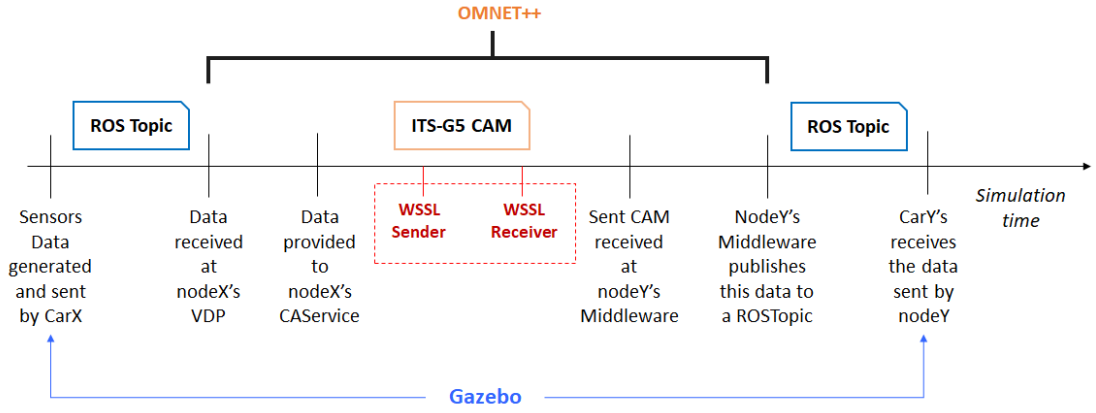


Figure 5.2: How WSSL was planned to be integrated with CopaDrive.

At last, CopaDrive demonstrated some requirements regarding the Operating System and the Omnetpp version installed. It only worked under Ubuntu 18.04 with Omnetpp version 5. Ubuntu 18 presents limitations regarding the C++ compiler, even requiring modifications inside the library and compilation flags to allow WSSL to work correctly. Also, bugs in the functionality of Visual Studio Code and other workspaces make it more complex to understand the errors and require more time to understand the sophisticated system functionalities. These and other issues made the integration between CopaDrive and WSSL impractical for now.

5.2 WSSL and ADACORSA

In this work, WSSL was integrated into ADACORSA implementation in the telemetry messages between a simulated drone and a GCS. The software regarding the Authority Handover and the communication between a drone and a GCS, happening via sockets, is developed primarily on C++, facilitating the integration with WSSL.

Moreover, other relevant works were already developed on drone communication and simulation frameworks inside ADACORSA, showing the viability of integrating safety and security methods in such systems. For instance, the work (Silva Borges 2021) presents solutions in terms of security by adding encryption to the communication protocols in the Handover processes between two trusted GCS. The work (Lopes 2021) proposes a drone simulation testbed capable of offering an authority transfer between GCS mid-flight. Last, work (Monteiro 2021) and (Gomes 2021) present a UAV-Based Safety Layer Architecture for the drone and the GCS point of view, respectively.

The drone telemetry is defined inside the Handover software as a struct containing several helpful information for UAVs's, such as the drone battery level in volts, longitude and latitude degree, altitude in meters, pitch, roll and yaw in rad per seconds, and velocity in different directions: north, east and down. This struct is illustrated in Listing 5.1.

```
1  struct telemetry_ho{
2      float battery_v;
3      double lat_deg;
4      double long_deg;
5      float altitude_m;
6      float pitch_rad_s;
7      float roll_rad_s;
8      float yaw_rad_s;
9      float vel_north_m_s;
10     float vel_east_m_s;
11     float vel_down_m_s;
12 };
```

Listing 5.1: Definition of the drone telemetry struct.

In this work, the GCS was created in a simulated environment using the QGroundControl, the simulation tool, and running connected to Ardupilot, the firmware. This firmware is also intended to be installed in the physical drone. For the tests, two desktops were connected via socket using an Ethernet cable, similar to the MQTT setup, and the tests were made with the simulation running. The QGroundControl application was updated with a simple mission to fly the drone from point A to point B, and the values of the drone telemetry were sent to the GCS with different frequencies. Figure 5.3 illustrates the setup used in the evaluation tests.

This thesis will focus on two main emergency situations: the system's behavior in case of a network failure and the system's behavior when receiving a malicious message. In the first scenario, WSSL helps to detect a network failure by monitoring the inter-message delay and sending these values to the GCS. In the second scenario, it detects possible malicious messages by tracking the message's status and protecting the system by discarding invalid ones.

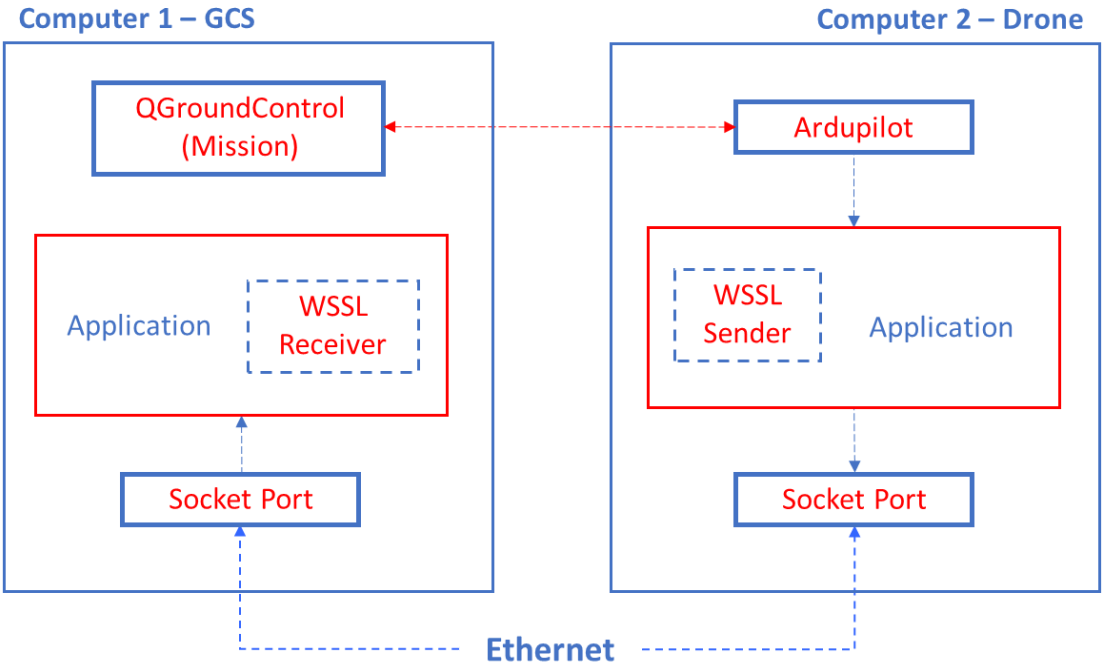


Figure 5.3: Laboratory setup for evaluating WSSL in the Handover code.

In case of a network failure, the GCS is responsible for checking the network parameters, so if the inter-message delay increases and surpasses an acceptable value, the GCS will detect that something is wrong with the connection and send an emergency command to the drone. The emergency command defined in this case is **returnToLaunch**, meaning the drone must return to the launch point and land. The system behavior in this emergency scenario and how the WSSL is integrated within the communication is detailed in a sequence diagram in Fig. 5.7

The system's behavior when receiving a malicious message will be tested and explained in Section 5.2.2, with the tests about error detection.

5.2.1 WSSL Cost

WSSL costs were calculated by sending a thousand, ten thousand, fifty thousand, and a hundred thousand messages, respectively. Similarly to the tests using MQTT, each one of the tests was also repeated for different frequencies.

Later tests showed that the safety layer does not have a significant increase in the overhead and that the security layer is responsible for most of the costs, having similar results as using WSSL with both layers. Thus, to save time, the individual costs for each of WSSL layers were only measured for lower amounts of messages.

Moreover, higher frequencies, such as 10000 Hz, already demonstrated unsatisfactory results. So, with that in mind and aiming to improve the tests, besides the frequencies used in the tests with MQTT, the tests were also made to 333 Hz. The results are presented in Figure 5.4.

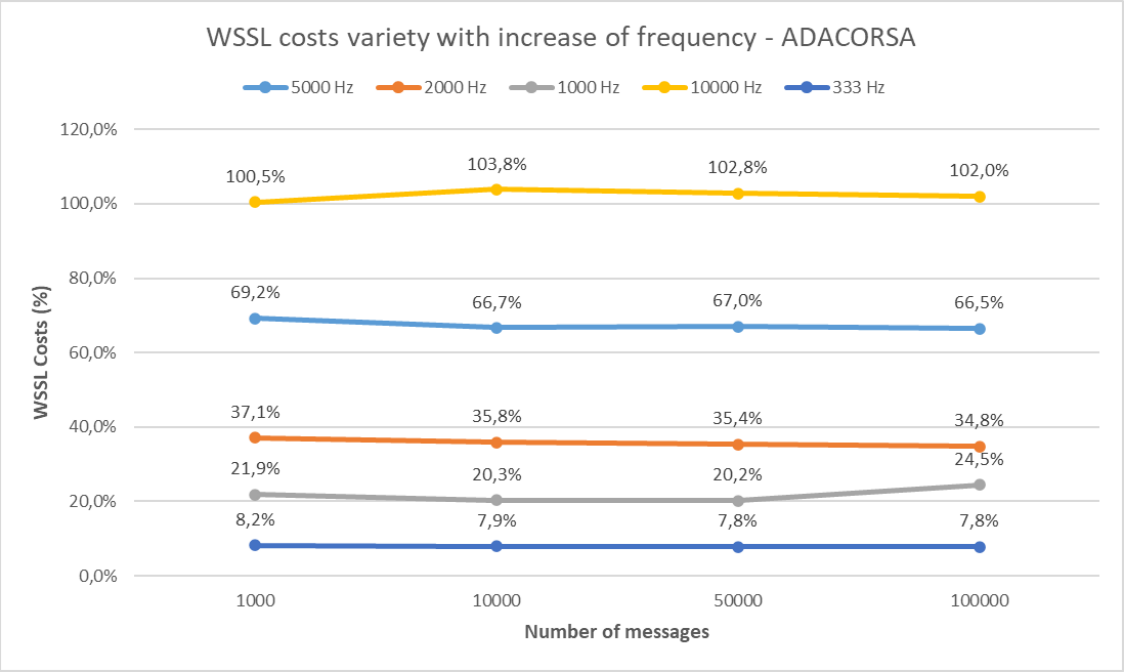


Figure 5.4: WSSL's costs in percentage for different frequencies - ADACORSA.

5.2.2 Error detection

The telemetry contains critical parameters of the drone, and one wrong parameter can change how the system will act next, which may cause a disruption in the service, emergency landing, and other unexpected behaviors. With that in mind, WSSL is responsible for discarding any malicious message or attempting to sabotage the system.

Therefore, the capability of WSSL to detect message errors must also be evaluated for the Handover simulation. For that, malicious messages were purposely sent to the GCS. Twenty messages were sent, and their velocity parameters heading north, east, and down were monitored. Then, two extra messages, messages 3 and 10, had their velocity parameter heading north (*vel_north_m_s*), purposely changed by a malicious sender to a dangerous velocity of 9 m/s.

The system velocity during sending messages without WSSL is plotted in Fig. 5.5. Since the system does not have any protection against malicious attacks, it is possible to see that the velocity parameter heading north (Vel. North) had a huge deviation because of the sabotaged message.

The system velocity using WSSL is plotted in Fig. 5.6. Notice that the malicious messages were successfully discarded, meaning that the system could continue working without significant differences and that the delay caused by the discarded messages is very low and would not damage the system's performance.

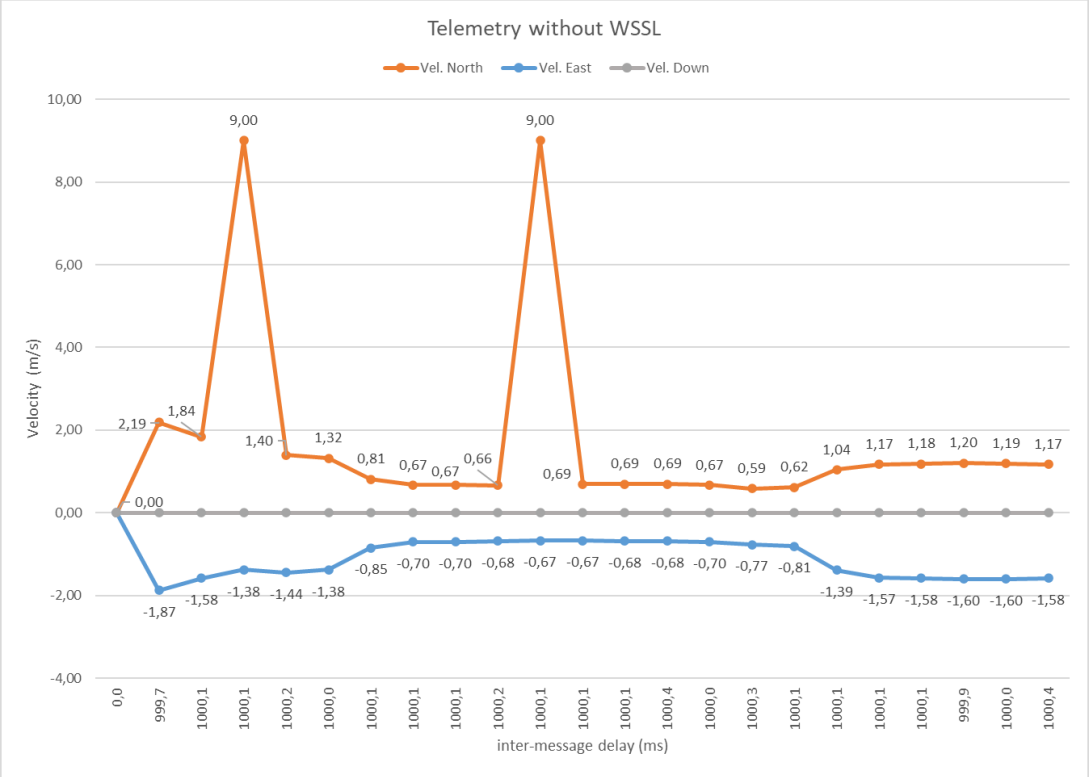


Figure 5.5: Telemetry velocities without WSSL.

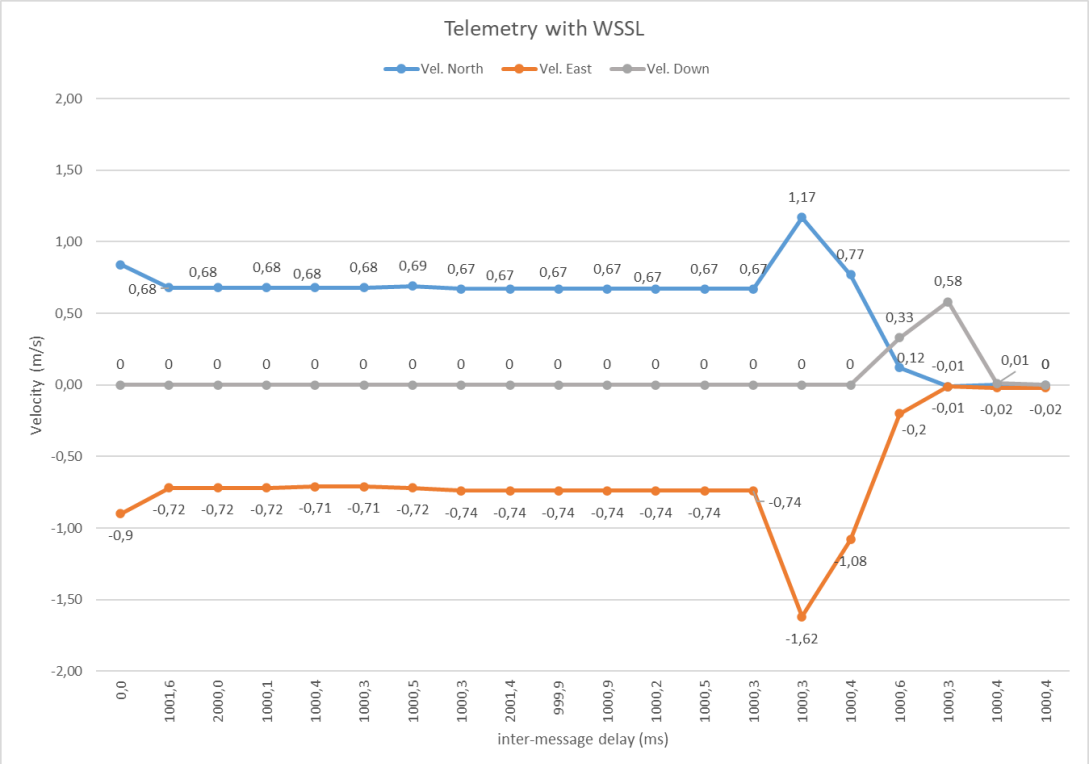


Figure 5.6: Telemetry velocities with WSSL.

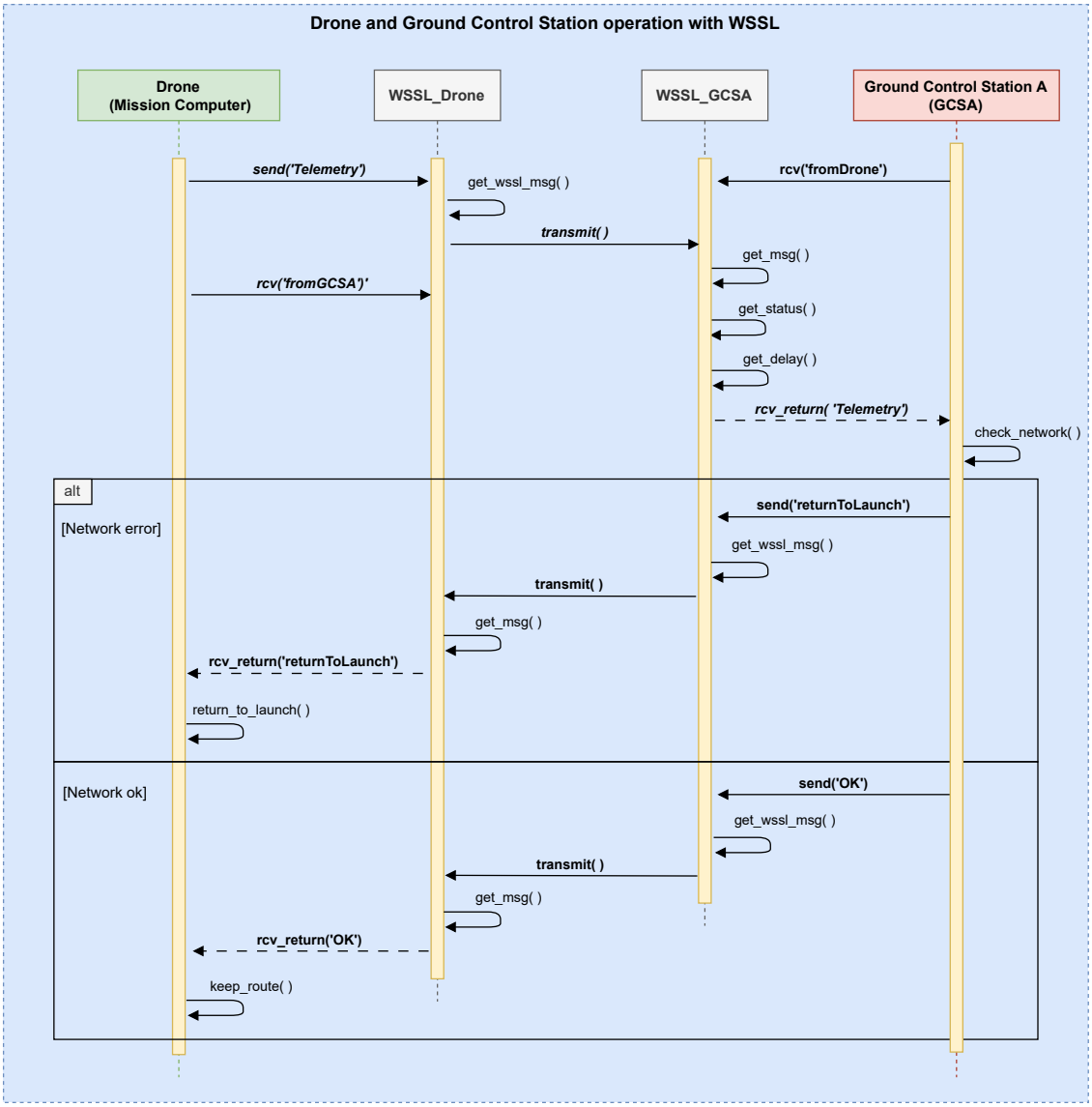


Figure 5.7: Sequence diagram illustrating the communication between Drone and GCS using WSSL.

5.3 Tests Results and Conclusions

In this chapter, the evaluation tests were based on sabotaging the network, analyzing the relation between increasing message frequency and message loss, and analyzing the results. Figure 5.8 presents charts with the performance of the system with and without WSSL. These charts illustrate the behavior and the system costs with the increase in the number of messages and frequency variation.

By sabotaging a message within the telemetry data and testing the system's behavior with and without WSSL, it was possible to assess the efficiency of WSSL in handling a malicious message. As shown in Fig. 5.5, the velocity suffered a deviation because of the sabotaged message when the system did not use WSSL. On the other hand, it was possible to see in Fig. 5.6 that malicious messages were successfully discarded without compromising the system's performance.

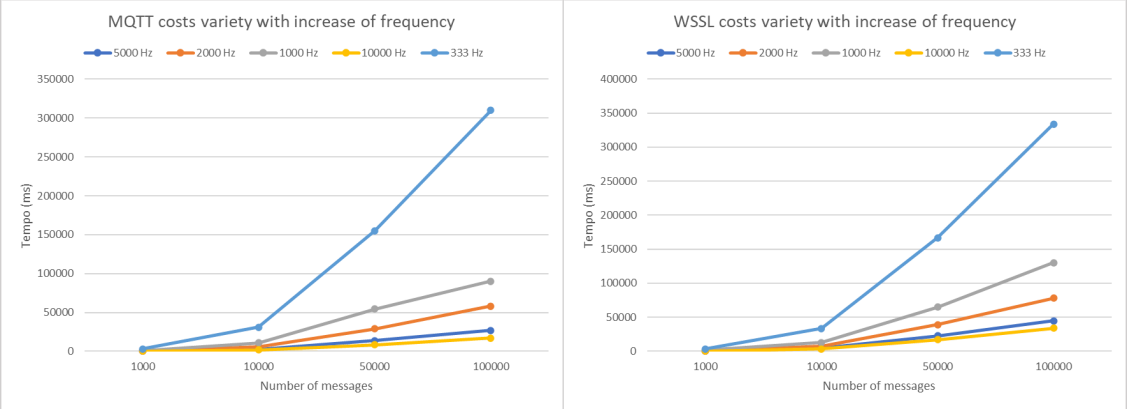


Figure 5.8: WSSL average costs versus frequency and number of messages when integrated with ADACORSA.

Regarding WSSL costs, it is possible to notice that for higher frequencies where the time was really short, the precision is not enough to show stable results, as it seeing using the high frequency of 10000 Hz where WSSL had costs of 100%. The costs start to be acceptable with a frequency of 2000 Hz, representing 35% of overhead. The best results are reached with the frequency of 333 Hz, where the costs are around 8%, and are stable even with a considerable amount of messages, such as a hundred thousand messages. Table 5.1 compares sending fifty messages with 1000 Hz and 333 Hz frequencies.

Table 5.1: Sent and reception time costs when sending fifty thousand msgs with the frequency of 1000 Hz and 333 Hz.

	1000 Hz				**333 Hz**			
	Without WSSL		**With WSSL**		**Without WSSL**		**With WSSL**	
Tests	**Sent time (ms)**	**Rcpt. time (ms)**	**Sent time (ms)**	**Rcpt. time (ms)**	**Sent time (ms)**	**Rcpt. time (ms)**	**Sent time (ms)**	**Rcpt. time (ms)**
T1	54150	54150	+11013	+11014	154813	154809	+12025	+12028
T2	54188	54187	+11037	+11039	154827	154823	+11945	+11947
T3	54204	54203	+10952	+10955	154829	154825	+11919	+11922
T4	54142	54141	+11023	+11024	154863	154858	+12020	+12024
T5	54199	54197	+11037	+11039	154827	154824	+12044	+12046
T6	54203	54202	+11010	+11013	154844	154840	+12046	+12049
T7	54330	54329	+10797	+10799	154840	154837	+12115	+12117
T8	54212	54210	+10799	+10802	154852	154849	+11969	+11971
T9	54146	54148	+10869	+10868	154834	154831	+12068	+12070
T10	54146	54148	+10869	+10868	154834	154831	+12068	+12070
Avg	**54197**	**54197**	**+10945**	**+10947**	**154834**	**154830**	**+12025**	**+12028**
%	**100,0**	**100,0**	**+20,2**	**+20,2**	**100,0**	**100,0**	**+7,8**	**+7,8**

The acceptable frequency for exchanging messages for UAVs still needs to be defined by standards since BVLOS is still not developed enough, and this kind of operation was still not allowed when this thesis was developed. The European standard (European Union Aviation

Safety Agency 2022) defines the rules and procedures for the operation of UAVs, but many challenges are still to be covered. So, to compare and analyze the results, we used the safety standards of other real-time applications, such as cooperative vehicles and avionics, as a basis. With that in mind, it is possible to say that WSSL fulfills safety requirements, working for higher frequencies and being promising for other real-time critical applications, such as the operation of UAVs.

At last, WSSL capability to detect the inter-message delay was also evaluated for this application. Equivalently to the previous chapter, a hundred messages were sent with a frequency of 10000 Hz, and the multiples of eleven were delayed by 11 ms. Figure 5.9 shows that WSSL can also successfully detect the delay in this application.

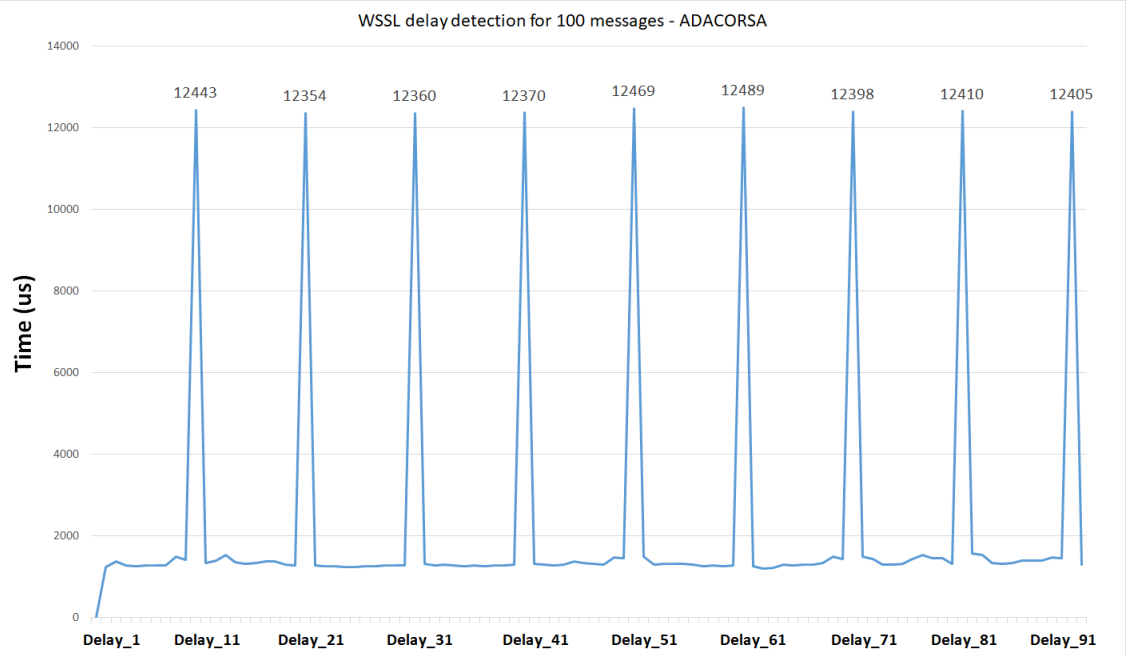


Figure 5.9: WSSL's inter-message delay detection in ADACORSA.

Chapter 6

Conclusions

This thesis describes the architecture and the implementation of a Wireless Safety and Security Layer (WSSL) to improve the applicability of Cyber-Physical Systems (CPS) devices by covering the main measures to detect possible faults and threads or keeping the error probability under a specific limit. WSSL aims to fill the necessity of practical applications related to Critical CPS devices and mitigate problems that must be urgently addressed in unsecured and unsafe wireless communication systems.

An extensive overview and analysis of the literature were necessary to develop this thesis. First, safety and security concepts had to be defined, and their impact on CPS had to be investigated. Next, a survey was conducted about the various challenges regarding CPS design, implementation, and communication between devices, including hard real-time and safety-critical systems. This work also examined the communication protocols related to CPS, their differences, limitations, and applications, analyzing how to increase safety and security on both ends of the communication between CPS. Finally, the existing solutions found in the literature were examined, defining strengths and weaknesses to define and justify WSSL's fundamentals.

This work's contributions are divided into three main aspects. First, according to the increased necessity to integrate CPS security in the safety domain, this thesis introduced the WSSL definition, describing its architecture, expected behavior, and essential concepts. A tutorial guide explaining how to install and use the library is also included, aiming to ease the integration of WSSL in future works.

Second, we evaluate the impact of WSSL on the data network in a controlled environment using MQTT. Implementing WSSL over a commonly used protocol for IoT applications such as MQTT reinforces its flexible character for other protocols and possible use in different conditions. Furthermore, the evaluation tests proved that the implementation costs of using WSSL do not imply high computational costs to the system and do not significantly increase the processing time for considerably high frequencies. Likewise, it was possible to demonstrate the ability of WSSL to successfully detect attack actions on the CPS, monitor network problems, and report them to the application.

Third, in a simulation environment, the evaluation tests when integrating WSSL with UAVs telemetry proved that the implementation costs of using WSSL were once more acceptable, even for considerable amounts of messages. Additionally, by monitoring the system inter-message delay efficiently, WSSL can be a safety ally in emergencies such as a network failure. Finally, It was also demonstrated that WSSL can detect and discard malicious messages, increasing the system's security.

Future research must focus on the enhancement of the safety and security layers. For example, a cryptography method could improve the security layer, increasing WSSL dependability. Also, the public-key signature could be evaluated regarding the performance, comparing the impact of different key sizes and signature sizes. There is also space to add parallelism to the code, which at this moment is entirely sequential, and test the impact of multithreading for cryptography.

Finally, future work should aim to integrate and evaluate WSSL in different projects that would allow a transition from the simulation to a real application, where WSSL could be proved efficient to supply security and safety in a practical way for CPS. Thus, the safety layer methods used for hazard identification may be tested in different environments, not only in simulators but in real applications, such as CPS applications implemented in aerial vehicles, cars, residential automation, and others. For instance, integrating WSSL into CopaDrive could change the paradigms of the project since quickly detecting a delay, message loss, or a message out-of-order in the platooning can be essential to avoid an accident. Thus, WSSL would be proved as a vital approach capable of guaranteeing vehicle connection and protecting communication.

Bibliography

ADACORSA (2023). *Airborne Data Collection on Resilient System Architectures*. Accessed 17 January 2023. url: https://adacorsa.eu/.

Aijaz, Adnan and A. Hamid Aghvami (2015). "Cognitive Machine-to-Machine Communications for Internet-of-Things: A Protocol Stack Perspective". In: *IEEE Internet of Things Journal* 2.2, pp. 103–112. doi: 10.1109/JIOT.2015.2390775.

Akerberg, Johan et al. (Sept. 2011). "Efficient integration of secure and safety critical industrial wireless sensor networks". In: *EURASIP Journal on Wireless Communications and Networking* 2011, pp. 1–13. doi: 10.1186/1687-1499-2011-100. url: https://doi.org/10.1186/1687-1499-2011-100.

Alguliyev, Rasim, Yadigar Imamverdiyev, and Lyudmila Sukhostat (2018). "Cyber-physical systems and their security issues". In: *Computers in Industry* 100, pp. 212–223. issn: 0166-3615. doi: https://doi.org/10.1016/j.compind.2018.04.017. url: https://www.sciencedirect.com/science/article/pii/S0166361517304244.

Analytics Market Research (Apr. 2023). *Cyber Physical System Market Consumption Analysis, Business Overview and Upcoming Trends 2032*. url: https://www.openpr.com/news/2870080/cyber-physical-system-market-consumption-analysis-business.

ArduPilot (July 2023). *ArduPilot Documentation*. Accessed 10 July 2023. url: https://ardupilot.org/ardupilot/.

Asplund, Fredrik et al. (2019). "Rapid Integration of CPS Security and Safety". In: *IEEE Embedded Systems Letters* 11.4, pp. 111–114. doi: 10.1109/LES.2018.2879631. url: https://doi.org/10.1109/LES.2018.2879631.

Atlam, Hany F. and Gary B. Wills (July 2020). "IoT Security, Privacy, Safety and Ethics". In: *Digital Twin Technologies and Smart Cities*. 2nd. Internet of Things. Cham: Springer International Publishing, pp. 123–149. isbn: 978-3-030-18732-3. doi: 10.1007/978-3-030-18732-3_8. url: https://doi.org/10.1007/978-3-030-18732-3_8.

Baheti, Radhakisan Sohanlal and Helen Gill (Mar. 2019). "Cyber-Physical Systems". In: *2019 IEEE International Conference on Mechatronics (ICM)*.

Balador, Ali et al. (Nov. 2018). "Wireless Communication Technologies for Safe Cooperative Cyber Physical Systems". In: *Sensors* 18. doi: 10.3390/s18114075. url: https://doi.org/10.3390/s18114075.

Bilenko, Pavel et al. (2020). "Evaluation of investment efficiency in cyber-physical systems and technologies in the construction industry". In: *IOP Conference Series: Materials Science and Engineering* 869. doi: 10.1088/1757-899X/869/6/062019. url: https://iopscience.iop.org/article/10.1088/1757-899X/869/6/062019.

Carreras Guzman, Nelson H., Igor Kozine, and Mary Ann Lundteigen (Dec. 2021). "An integrated safety and security analysis for cyber-physical harm scenarios". In: *Safety Science* 144. doi: 10.1016/j.ssci.2021.105458. url: https://doi.org/10.1016/j.ssci.2021.105458.

Cecchetti, Gabriele et al. (2013). "An Implementation of EURORADIO Protocol for ERTMS Systems". In: *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering* 7.6, pp. 693–702.

Chen, Li-jie et al. (2011). "Performance analysis and verification of safety communication protocol in train control system". In: *Computer Standards & Interfaces* 33.5, pp. 505–518. doi: 10.1016/j.csi.2011.02.006. url: https://doi.org/10.1016/j.csi.2011.02.006.

Coopmans, Calvin et al. (Jan. 2015). "Cyber-Physical Systems Enabled by Small Unmanned Aerial Vehicles". In: 2nd. Chicago: Springer Netherlands, pp. 2835–2860. isbn: 978-90-481-9706-4. doi: 10.1007/978-90-481-9707-1_106. url: https://doi.org/10.1007/978-90-481-9707-1_106.

Cppreferences (2022). *Date and time utilities*. Accessed 08 June 2022. url: https://en.cppreference.com/w/cpp/chrono.

Cunha Rocha, Marcia et al. (2023). "A WSSL Implementation for Critical Cyber-Physical Systems Applications". In: *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*. CPS-IoT Week '23. San Antonio, TX, USA: Association for Computing Machinery, pp. 192–197. doi: 10.1145/3576914.3587507. url: https://doi.org/10.1145/3576914.3587507.

Eclipse (2022). *Eclipse Mosquitto homepage*. Accessed 19 January 2023. url: https://mosquitto.org/.

Ell, Maddy and Robbie Gallucci (July 2022). *Official Statistics. Cyber Security Breaches Survey 2022*. url: https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2022/cyber-security-breaches-survey-2022.

EN 50159 (2010). *Railway applications. Communication, signaling and processing systems. Safety-related communication in transmission systems*. European Standards. url: https://www.en-standard.eu/ilnas-en-50159-railway-applications-communication-signalling-and-processing-systems-safety-related-communication-in-transmission-systems/.

Esterle, Lukas and Radu Grosu (Nov. 2016). "Cyber-physical systems: challenge of the 21st century". In: *e & i Elektrotechnik und Informationstechnik* 133, pp. 299–303. doi: 10.1007/s00502-016-0426-6. url: https://doi.org/10.1007/s00502-016-0426-6.

ETSI EN 302 663 (Nov. 2012). *ETSI EN 302 663 V1.2.0. Intelligent Transport Systems (ITS); Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band*. Tech. rep. V1.2.0. European Telecommunications Standards Institute. url: https://www.etsi.org/deliver/etsi_en/302600_302699/302663/01.02.00_20/en_302663v010200a.pdf.

ETSI TR 102 638 (June 2009). *ETSI TR 102 638 V1.1.1. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions*. Tech. rep. V1.1.1. European Telecommunications Standards Institute. url: https://www.etsi.org/deliver/etsi_tr/102600_102699/102638/01.01.01_60/tr_102638v010101p.pdf.

European Global Navigation Satellite Systems (EGNSS) for drones operations: white paper (2020). Publications Office. doi: doi/10.2878/52219.

European Union Aviation Safety Agency (Sept. 2022). *Easy Access Rules for Unmanned Aircraft Systems (Regulations (EU) 2019/947 and 2019/945)*. Tech. rep. V1.3.1. European Union Aviation Safety Agency (EASA). url: https://www.easa.europa.eu/en/document-library/easy-access-rules/easy-access-rules-unmanned-aircraft-systems-regulations-eu.

Filho, Enio Vasconcelos, Nuno Guedes, et al. (Apr. 2020). "Towards a Cooperative Robotic Platooning Testbed". In: *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), 2020*. Ponta Delgada, Portugal: IEEE, pp. 332–337. isbn: 978-1-72817-078-7. doi: 10.1109/ICARSC49921.2020.9096132. url: https://doi.org/10.1109/ICARSC49921.2020.9096132.

Filho, Enio Vasconcelos, Ricardo Severino, Anis Koubaa, et al. (June 2021). "A Wireless Safety and Security Layer Architecture for Reliable Co-CPS". In: *DCE21- Symposium on Electrical and Computer Engineering: Book of Abstracts*. Vol. 1. Porto, Portugal: FEUP. isbn: 978-972-752-276-7.

Filho, Enio Vasconcelos, Ricardo Severino, Joao Rodrigues, et al. (July 2021). "CopaDrive: An Integrated ROS Cooperative Driving Test and Validation Framework". In: *Robot Operating System (ROS)*. Vol. 962. Studies in Computational Intelligence. Cham: Springer International Publishing, pp. 121–174. isbn: 978-3-030-75472-3. doi: 10.1007/978-3-030-75472-3_4. url: http://link.springer.com/10.1007/978-3-030-75472-3_4.

FLY-PT (2023). *Mobilize the national aviation industry to transform the future urban air transport*. Accessed 28 July 2023. url: http://flypt.pt/.

Gódor, Győző et al. (2015). "A survey of handover management in LTE-based multi-tier femtocell networks: Requirements, challenges and solutions". In: *Computer Networks* 76, pp. 17–41. issn: 1389-1286. doi: https://doi.org/10.1016/j.comnet.2014.10.016. url: https://www.sciencedirect.com/science/article/pii/S1389128614003715.

Gomes, Filipe Manuel Moura (Sept. 2021). "UAV-Based Safety Layer Architecture: A Control Station Point Of View". In: Bachelor's dissertation.

Greer, Christopher et al. (Mar. 2019). "Cyber-Physical Systems and Internet of Things". en. In: doi: https://doi.org/10.6028/NIST.SP.1900-202.

Heinrich, Markus et al. (July 2019). "Security Requirements Engineering in Safety-Critical Railway Signalling Networks". In: *Security and Communication Networks* 2019, pp. 1–14. doi: 10.1155/2019/8348925. url: https://doi.org/10.1155/2019/8348925.

Hermann, Mario, Tobias Pentek, and Boris Otto (2016). "Design Principles for Industrie 4.0 Scenarios". In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. Koloa, HI, USA: IEEE, pp. 3928–3937. isbn: 978-0-7695-5670-3. doi: 10.1109/HICSS.2016.488. url: https://doi.org/10.1109/HICSS.2016.488.

Hoffmann, Javier et al. (2018). "Towards a Safety and Energy Aware protocol for Wireless Communication". In: *2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. Lille, France: IEEE, pp. 1–6. isbn: 978-1-5386-7957-9. doi: 10.1109/ReCoSoC.2018.8449380. url: https://doi.org/10.1109/ReCoSoC.2018.8449380.

IEC 61508 (2010). *Functional safety for electrical/electronic/programmable electronic systems*. 2nd. International Electrotechnical Commission. url: https://www.61508.org/index.php.

Indira, N., S. Rukmanidevi, and A.V. Kalpana (2020). "Light Weight Proactive Padding Based Crypto Security System in Distributed Cloud Environment:" en. In: *International Journal of Computational Intelligence Systems* 13.1, p. 36. issn: 1875-6883. doi: 10.2991/ijcis.d.200110.001. url: https://www.atlantis-press.com/article/125931996.

ISO/IEC 27001:2013 (2013). *Information technology — Security techniques — Information security management systems — Requirements*. ISO. url: https://www.iso.org/standard/54534.html.

Ji, Zuzhen et al. (Apr. 2021). "Harmonizing safety and security risk analysis and prevention in cyber-physical systems". In: *Process Safety and Environmental Protection* 148, pp. 156–

178. doi: 10.1016/j.psep.2021.03.004. url: https://doi.org/10.1016/j.psep.2021.03.004.

Johnson, Chris (2012). "CyberSafety: CyberSecurity and Safety-Critical Software Engineering". In: *Achieving Systems Safety*. Ed. by Chris Dale and Tom Anderson. London: Springer London, pp. 85–95. isbn: 978-1-4471-2494-8. url: https://doi.org/10.1007/978-1-4471-2494-8_8.

Kabir, Sohag (2021). "Internet of Things and Safety Assurance of Cooperative Cyber-Physical Systems: Opportunities and Challenges". In: *IEEE Internet of Things Magazine* 4.2, pp. 74–78. doi: 10.1109/IOTM.0001.2000062.

Kavallieratos, Georgios, Sokratis Katsikas, and Vasileios Gkioulos (Apr. 2020). "Cybersecurity and Safety Co-Engineering of Cyberphysical Systems—A Comprehensive Survey". In: *Future Internet* 12, pp. 505–518. doi: 10.3390/fi12040065. url: https://doi.org/10.3390/fi12040065.

Kriaa, Siwar et al. (July 2015). "A survey of approaches combining safety and security for industrial control systems". In: *Reliability Engineering & System Safety* 139, pp. 156–178. doi: 10.1016/j.ress.2015.02.008. url: https://doi.org/10.1016/j.ress.2015.02.008.

Li, Kai et al. (Apr. 2020). "Design and Implementation of Secret Key Agreement for Platoon-based Vehicular Cyber-physical Systems". In: *ACM Transactions on Cyber-Physical Systems* 4.2, pp. 1–20. issn: 2378-962X, 2378-9638. doi: 10.1145/3365996. url: https://doi.org/10.1145/3365996.

Libsodium (2022). *Introduction - Libsodium*. Accessed 07 September 2022. url: https://doc.libsodium.org/.

Lind, Oskar (2020). "Defending against denial of service attacks in ETSI ITS-G5 networks (Master's thesis)". MA thesis. Linköping, SE: Linköpings universitet.

Lopes, Gustavo Castro (Sept. 2021). "Drone Handover: Missions Overview". In: Bachelor's dissertation.

Lyu, Xiaorong, Yulong Ding, and Shuang-Hua Yang (Apr. 2019). "Safety and Security Risk Assessment in Cyber-Physical Systems". In: *IET Cyber-Physical Systems: Theory & Applications* 4 (3). doi: 10.1049/iet-cps.2018.5068. url: https://doi.org/10.1049/iet-cps.2018.5068.

Magro, Micaela Caserza, Paolo Pinceti, and Luca Rocca (2016). "Can we use IEC 61850 for safety related functions?" In: *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*. Florence, Italy: IEEE. isbn: 978-1-5090-2320-2. doi: 10.1109/EEEIC.2016.7555402. url: https://doi.org/10.1109/EEEIC.2016.7555402.

Márcia, C. Rocha and Vasconcelos Filho Enio (Oct. 2022). *WSSL_Library*. Version 1.0.0. CISTER Research Centre. url: https://github.com/marciacr/WSSL_Library.

Marwedel, Peter (Jan. 2021). *Embedded System Design. Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. 4th. Embedded Systems. Chicago: Springer Cham, pp. 01–20. isbn: 978-3-030-60910-8. doi: 10.1007/978-3-030-60910-8. url: https://doi.org/10.1007/978-3-030-60910-8.

Mohammed, Abdalbasit and Nurhayat Varol (June 2019). "A Review Paper on Cryptography". In: pp. 1–6. doi: 10.1109/ISDFS.2019.8757514.

Monteiro, Stéphane Joaquim Lourenço (Sept. 2021). "UAV-Based Safety Layer Architecture: A Drone Point Of View". In: Bachelor's dissertation.

Moore, Susan (Sept. 2020). *Gartner Predicts 75% of CEOs Will be Personally Liable for Cyber-Physical Security Incidents by 2024*. url: https://www.gartner.com/en/

newsroom/press-releases/2020-09-01-gartner-predicts-75--of-ceos-will-be-personally-liabl.

MQTT (2022). *MQTT homepage*. Accessed 19 January 2023. url: https://mqtt.org/.

Munoz, Antonio and Antonio Mafia (2014). "Software and hardware certification techniques in a combined certification model". In: *2014 11th International Conference on Security and Cryptography (SECRYPT)*, pp. 1–6. isbn: 978-9-8985-6595-2.

Nandi, Giann Spilere, David Pereira, and José Proença (2021). *Cyber-Physical Systems – Addressing Safety and Security Aspects in the Presence of Runtime Monitors*. Accessed 22 July 2023. url: https://valu3s.eu/cyber-physical-systems-addressing-safety-and-security-aspects-in-the-presence-of-runtime-monitors/.

Navet, Nicolas and Stephan Merz, eds. (Mar. 2013). *Modeling and Verification of Real-time Systems: Formalisms and Software Tools*. 1st. Wiley-ISTE, p. 448. isbn: 978-1-118-62395-4.

Pivoto, Diego G.S. et al. (2021). "Cyber-physical systems architectures for industrial internet of things applications in Industry 4.0: A literature review". In: *Journal of Manufacturing Systems* 58, pp. 176–192. doi: 10.1016/j.jmsy.2020.11.017. url: https://doi.org/10.1016/j.jmsy.2020.11.017.

Politi, Elena et al. (2022). "The future of safe BVLOS drone operations with respect to system and service engineering". In: *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 133–140. doi: 10.1109/SOSE55356.2022.00022.

QGroundControl (2023). *QGroundControl User Guide*. Accessed 10 July 2023. url: https://docs.qgroundcontrol.com/master/en/index.html.

Rastocny, Karol et al. (Jan. 2016). "Modelling of hazards effect on safety integrity of open transmission systems". In: 35, pp. 470–496. url: https://www.cai.sk/ojs/index.php/cai/article/view/3232.

Rawat, Danda B. and Chandra Bajracharya (2017). *Vehicular Cyber Physical Systems. Adaptive Connectivity and Security*. 1st. USA: Springer Cham. isbn: 978-3-319-44494-9. doi: 10.1007/978-3-319-44494-9.

Riebl, Raphael (2020). *Artery. V2X Simulation Framework*. Accessed 29 July 2023. url: http://artery.v2x-research.eu/.

ROS - Robot Operating System (2022). Accessed 14 January 2023. url: https://www.ros.org/.

Schmittner, Christoph et al. (2016). "Using SAE J3061 for Automotive Security Requirement Engineering". In: *Computer Safety, Reliability, and Security*. Vol. 9923. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 157–170. isbn: 978-3-319-45480-1. doi: 10.1007/978-3-319-45480-1_13. url: http://link.springer.com/10.1007/978-3-319-45480-1_13.

Shallal, Qahtan and Mohammad Bokhari (Aug. 2016). "A Review on Symmetric Key Encryption Techniques in Cryptography". In: *International Journal of Computer Applications*, p. 43.

Shayea, Ibraheem et al. (2022). "Handover Management for Drones in Future Mobile Networks - A Survey". In: *Sensors* 22.17. issn: 1424-8220. doi: 10.3390/s22176424. url: https://www.mdpi.com/1424-8220/22/17/6424.

Silva, Diego R. C. et al. (June 2018). "Latency evaluation for MQTT and WebSocket Protocols: an Industry 4.0 perspective". en. In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. Natal: IEEE, pp. 01233–01238. isbn: 978-1-5386-6950-1. doi: 10.1109/ISCC.2018.8538692. url: https://ieeexplore.ieee.org/document/8538692/ (visited on 10/30/2022).

Silva Borges, Eduardo da (Sept. 2021). "Drone Testbed". In: Bachelor's dissertation.

Statista (Aug. 2022). *Cyber crime: number of compromises and victims in U.S. 2005-H1 2022*. url: https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/.

Törngren, Martin and Ulf Sellgren (July 2018). "Complexity Challenges in Development of Cyber-Physical Systems". In: ed. by Marten Lohstroh, Patricia Derler, and Marjan Sirjani. Vol. 10760, pp. 478–503. isbn: 978-3-319-95245-1. doi: 10.1007/978-3-319-95246-8_27. url: https://doi.org/10.1007/978-3-319-95246-8_27.

Vieira, Bruno et al. (2019). "COPADRIVe - A Realistic Simulation Framework for Cooperative Autonomous Driving Applications". In: *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*. Graz, Austria: IEEE, pp. 1–6. isbn: 978-1-7281-0142-2. doi: 10.1109/ICCVE45908.2019.8965161. url: https://doi.org/10.1109/ICCVE45908.2019.8965161.

Vinel, Alexey, Nikita Lyamin, and Pavel Isachenkov (2018). "Modeling of V2V Communications for C-ITS Safety Applications: A CPS Perspective". In: *IEEE Communications Letters* 22.8, pp. 1600–1603. doi: 10.1109/LCOMM.2018.2835484. url: https://doi.org/10.1109/LCOMM.2018.2835484.

Vortex-CoLab (2022). *WHO WE ARE, ORGANISATION*. Accessed 08 June 2022. url: https://www.vortex-colab.com/organisation/.

Wolf, Marilyn and Dimitrios Serpanos (2018). "Safety and Security in Cyber-Physical Systems and Internet-of-Things Systems". In: *Proceedings of the IEEE* 106.1, pp. 9–20. doi: 10.1109/JPROC.2017.2781198.

Xiao, Yang, Hsiao-Hwa Chen, et al. (Oct. 2006). "MAC Security and Security Overhead Analysis in the IEEE 802.15.4 Wireless Sensor Networks". In: *EURASIP Journal on Wireless Communications and Networking* 2006.2, p. 81. doi: 10.1155/WCN/2006/93830. url: https://doi.org/10.1155/WCN/2006/93830.

Xiao, Yang, S. Sethi, et al. (2005). "Security services and enhancements in the IEEE 802.15.4 wireless sensor networks". In: *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.* Vol. 3. St. Louis, MO, USA: IEEE, pp. 5–. isbn: 0-7803-9414-3. doi: 10.1109/GLOCOM.2005.1577958. url: https://doi.org/10.1109/GLOCOM.2005.1577958.

Yildiz, Melih et al. (May 2021). "Experimental Investigation of Communication Performance of Drones Used for Autonomous Car Track Tests". en. In: *Sustainability* 13.10, p. 5602. issn: 2071-1050. doi: 10.3390/su13105602. url: https://www.mdpi.com/2071-1050/13/10/5602 (visited on 10/30/2022).

Zhang, Dan et al. (Oct. 2021). "A survey on attack detection, estimation and control of industrial cyber-physical systems". In: *ISA Transactions* 116, pp. 1–16. doi: 10.1016/j.isatra.2021.01.036. url: https://doi.org/10.1016/j.isatra.2021.01.036.

Zhang, Junqing et al. (Aug. 2017). "Securing Wireless Communications of the Internet of Things from the Physical Layer, An Overview". In: *Entropy* 19. doi: 10.3390/e19080420. url: https://doi.org/10.3390/e19080420.

Zhou, Xiang-Yu et al. (Feb. 2021). "A system-theoretic approach to safety and security co-analysis of autonomous ships". In: *Ocean Engineering* 222. doi: 10.1016/j.oceaneng.2021.108569. url: https://doi.org/10.1016/j.oceaneng.2021.108569.

Chapter 7

Appendix A - WSSL Instalation Tutorial

7.1 How do I get set up?

This work was developed using Ubuntu version 20.04, so the command lines are based on Linux bash.

In this tutorial, we will cover the following:

- Environment configured with C++ compilators (g++);
- Install dependencies and third-party libraries;
- Install MQTT using libmosquitto broker;
- Build the library using Make and CMake;

7.1.1 Make and CMake

Install and build essential packages and libraries:

```
$ sudo apt install build-essential
```

To install Make in Ubuntu OS, use the following commands:

```
$ sudo apt install make
```

Check if Make is installed and update it to the last version:

```
$ sudo apt update make -version
```

WSSL requires cmake version 3.10. If the installation was successful, proceed to the next section.

7.1.2 CryptoIdentity library

Requires sodium library. To install it via the command line:

```
$ sudo apt install libsodium-dev
```

You may also install it via similar installers or through local building:

`https://libsodium.gitbook.io/doc/installation`

The next step is going to the location of WSSL and including on your project build files the CryptoLib and sodium to the *CMakeList.txt* file:

```
cmake:
add_executable(<executable-name> ${HEADERS} <file.
    cpp>)
target_link_libraries(<project name> CryptoLib
    sodium)
```

To build the library, use the following command:

```
$ cmake -Bbuild -H. && cmake --build build
```

7.1.3 Using WSSL with MQTT Libmosquitto

First, install libmosquitto:

```
$ sudo apt-add-repository ppa:mosquitto-dev/
    mosquitto-ppa
$ sudo apt-get update sudo apt-get install
    libmosquitto-dev
```

Then, install mosquitto clients:

```
$ sudo apt-get install mosquitto
$ sudo apt-get install mosquitto-clients
$ sudo apt clean
```

Include the library in your code:

```
#include <mosquitto.h>
```

Create a custom configuration file inside mosquitto path and modify it:

```
$ sudo nano /etc/mosquitto/conf.d/custom.conf
```

Change the default listener port from 1883 to 1885 and allow anonymous authentication by adding the following lines:

```
listener 1885
allow_anonymous true
```

Restart the service broker:

```
$ service mosquitto restart
```

Check if the broker status is running:

```
$ service mosquitto status
```

You can now run your code using libmosquitto. More information can be found in the following link: `https://mosquitto.org/download/`