A WSSL Implementation for Critical Cyber-Physical Systems Applications

Márcia Rocha*

Enio Filho*

Fernando Alves

Sérgio Penna*

Pedro Miguel Santos*

Eduardo Tovar*

*CISTER Research Centre

CISTER-TR-230504

2023/05/10

A WSSL Implementation for Critical Cyber-Physical Systems Applications

Márcia Rocha*, Enio Filho*, Fernando Alves, Sérgio Penna*, Pedro Miguel Santos*, Eduardo Tovar*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: rocha@isep.ipp.pt, enpvf@isep.ipp.pt, fernando.alves@vortex-colab.com, sdp@isep.ipp.pt, pss@isep.ipp.pt, emt@isep.ipp.pt

https://www.cister-labs.pt

Abstract

The advancements in wireless communication technologies have enabled unprecedented pervasiveness and ubiquity of Cyber-Physical Systems (CPS). Such technologies can now empower true Systemsof-Systems, which cooperate to achieve more complex and efficient functionalities. However, for CPS applications to become a reality, safety and security must be guaranteed, particularly in critical systems, since they rely on open communication systems prone to intentional and non-intentional interferences. We propose designing a Wireless Safety and Security Layer (WSSL) architecture to be implemented in critical CPS applications to address these issues. WSSL increases the reliability of these critical communications by enabling the detection of communication errors. Furthermore, it increases the CPS security using a message signature process that uniquely identifies the sender. So, we present the WSSL architecture and its implementation over an MQTT protocol. We prove that WSSL does not significantly increase the system transmission costs and demonstrate its capability to ensure safety and security, allowing it to be used in any general or critical CPS.

A WSSL Implementation for Critical Cyber-Physical Systems Applications

Marcia Cunha Rocha
rocha@isep.ipp.pt
CISTER Research Centre, ISEP
Porto, Portugal

Enio Vasconcelos Filho
enpvf@isep.ipp.pt
CISTER Research Centre, ISEP
Porto, Portugal

Fernando Alves
fernando.alves@vortex-colab.com
Vortex CoLab
Vila Nova de Gaia, Portugal

Sergio Penna
sdp@isep.ipp.pt
CISTER Research Centre, ISEP
Porto, Portugal

Pedro M. Santos
pss@isep.ipp.pt
CISTER Research Centre, ISEP
Porto, Portugal

Eduardo Tovar
emt@isep.ipp.pt
CISTER Research Centre, ISEP
Porto, Portugal

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, which cooperate to achieve more complex and efficient functionalities. However, for CPS applications to become a reality, safety and security must be guaranteed, particularly in critical systems, since they rely on open communication systems prone to intentional and non-intentional interferences. We propose designing a Wireless Safety and Security Layer (WSSL) architecture to be implemented in critical CPS applications to address these issues. WSSL increases the reliability of these critical communications by enabling the detection of communication errors. Furthermore, it increases the CPS security using a message signature process that uniquely identifies the sender. So, we present the WSSL architecture and its implementation over an MQTT protocol. We prove that WSSL does not significantly increase the system transmission costs and demonstrate its capability to ensure safety and security, allowing it to be used in any general or critical CPS.

CCS CONCEPTS

• **Computer systems organization → Reliability**; • **Networks → Cyber-physical networks**; **Error detection and error correction**; **Security protocols**.

KEYWORDS

Safety, Security, Cyber-Physical Systems.

ACM Reference Format:
Marcia Cunha Rocha, Enio Vasconcelos Filho, Fernando Alves, Sergio Penna, Pedro M. Santos, and Eduardo Tovar. 2023. A WSSL Implementation for Critical Cyber-Physical Systems Applications. In *Cyber-Physical Systems and Internet of Things Week 2023 (CPS-IoT Week Workshops '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3576914.3587507

CPS-IoT Week Workshops '23, May 09–12, 2023, San Antonio, TX, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0049-1/23/05.
https://doi.org/10.1145/3576914.3587507

1 INTRODUCTION

Cyber-Physical Systems (CPS), whether cooperative or not, are mainly subject to malicious agents. Furthermore, their implementation based on devices from different manufacturers allows security flaws [8]. Accordingly, EN 50159 proposes an end-to-end safety approach using the black channel principle [11]. In this approach [22], 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.

Several studies have been performed on the safety and security of CPS devices and how often they influence each other [6, 15, 25]. Still, only a few address both of them in a practical way. According to [17], 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.

On the other hand, just a few studies comprise safety. Usually, they are focused on a theoretical approach [4], like surveys analyzing existing methods [13] or suggesting the importance of safety in CPS devices [3]. On the other hand, some complex systems are being developed: a risk assessment framework focused on quick and guided feedback about safety and security [2], or a cyber-physical system analytical framework [23], where, despite their importance, neither of them provide operational tools.

Safety standards like the IEC 61508 define several aspects and models to increase systems safety but still need to advance to real applications [19]. Furthermore, most of the works encompassing safety are closed in a specific application field, which reduces their applicability to general CPS, like EURORADIO [7, 16], WirelessHART [1], ISO 26262 and SAE J3061 [20]. Unfortunately, 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. Consequently, integrating safety and security remains a challenge of great importance.

In this work, we present a modular Wireless Safety and Security 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 [12] and real environments [10]. Moreover, although some defenses involve verifying the origin and destination of the messages sent, 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 three 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 MQTT as the communication protocol between two devices, numerically analyzing the impact of its use on the data network.
- Demonstrate the ability of WSSL to detect attack actions on the CPS, monitor network problems, and report them to the application, increasing the application's security and safety.

This paper is divided as follows. Section 2 presents the WSSL definition. The WSSL architecture is explained in Section 3, while the WSSL protocols and a deeper view of the system, including how the sender and the receiver work, are described in Section 4. In Section 5, evaluation tests and their results are demonstrated for different use cases of WSSL. Finally, conclusions and future works are presented in Section 6.

2 WSSL DEFINITION

The proposed WSSL consists of an additional layer to the adopted communication system, implementing 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. Its use seeks to increase trust between the sender and receiver since communication failures or malicious interactions can have critical consequences. It can be used in open communication systems, where transmissions is unsafe. The implementation is agnostic to the used communication protocols, thus being generically applicable to many use-cases, as illustrated in Fig. 1.

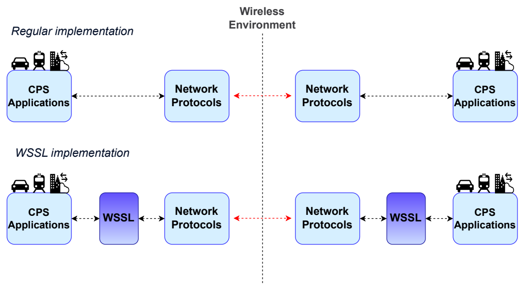


Figure 1: Basic implementation of WSSL

The primary function of the WSSL proposed in this work is the detection of communication issues between CPS devices, whether or not caused by malicious agents. These issues can be **message repetition**, **packet loss**, or **inter-message delay**. By definition, we assume this 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. This signature model guarantees the integrity of the received messages, discarding those with data loss.

Considering the diversity of CPS devices, WSSL was developed as a static library compiled in C++. It can be attached to most systems on specific hardware or as part of the original system. Thus, its implementation cost is minimal, and its benefits significant, allowing its application in low-cost or high-performance systems. Although it detects the conditions and problems of the communications network, WSSL does not alter the operation of the device it was implemented in, informing the application about the detection of the event so that the device can handle it. In this model, 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 SD's, depending on the processing capability, and can also be an SD to other devices. The general flow of the WSSL architecture is illustrated in Fig. 2.

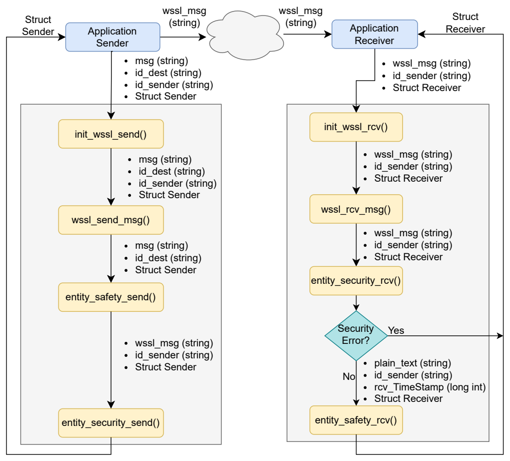


Figure 2: General flow of the WSSL Library

WSSL library implementation codes are open-source and are available on GitHub repository [5].

3 WSSL ARCHITECTURE

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 must provide the message content (*msg*), the *RD_ID*, and the *SD_ID* to the *WSSL_Sender*. Conversely, the RD must provide the signed message (*signed_msg*) and the *SD_ID* to retrieve the data from the *WSSL_Receiver*. 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 by both SD and RD, including: the last message of each sender, their timestamp, sequence number, communication status, and inter-message delay.

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 *safe_msg*. Then, the *safe_msg* is transmitted to the *WSSL_Sender* security layer that signs it, adding information about the key and the message size, returning a *signed_msg* to the application in the SD. In the proposed architecture, the SD is responsible for sending the *signed_msg* as an ordinary message. This way, WSSL does not interfere with the device's communication protocol.

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

After retrieving the safety data from the *safe_msg*, 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 and delay between the sender and receiver. These delays are calculated if the message is not out-of-order or a duplication. If the delay value exceeds a threshold, the *WSSL_Receiver* will warn the application.

4 WSSL COMPONENTS

WSSL is divided in the Sender and the Receiver sides. Each side has a security and a safety entity function responsible for implementing the WSSL itself, and its own packet format: *Struct Sender*, in case of the *WSSL_Sender*, or *Struct Receiver*, 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.

4.1 WSSL Sender

The SD application calls the *WSSL_Sender*, through the function *init_wssl_send*, passing the objects: *msg*, *SD_ID*, *RD_ID*, and the *Struct Sender*. First, the *msg* passes through the safety function, *entity_safety_send*, where the safety features are added. Then, the *safe_msg*, the *RD_ID*, and the *Struct Sender* are passed to the security function, *entity_security_send*. Finally, the *safe_msg* is signed and stored inside the *Struct Sender* to be returned to the *WSSL_Sender* application. The *safe_msg* is a concatenated string with the following format:

Time Stamp / Message data / Sequence Number

4.1.1 *WSSL Sender Entity for Safety.*

The safety entity adds a timestamp and a sequence number to the *msg*, creating and updating the *WSSL_Sender*'s table. In addition, it returns the *Struct Sender* to be used in the security entity. The timestamp in microseconds is obtained using C++ Chrono library, a flexible collection of types that track time with varying degrees of precision.

The *Struct Sender* 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. A 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 *Struct Sender* 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 structure, waiting for the next message. Finally, the *Struct Sender* is returned to be used in the *WSSL_Sender*'s security entity, containing the *safe_msg*.

4.1.2 *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. In this work, we assume the public keys were safely exchanged between the identities using a safe connection, using the protocol described in [18]. These keys are used to sign and remove the message signatures, ensuring message integrity and authenticity. The method *sign* adds the signature to the *safe_msg*, creating the *signed_msg*. 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 [14]. In the same way, as the *WSSL_Receiver* will need the message size, the Security Entity adds to the *signed_msg* the *signed_msg* size (*fullSize*) and the *SD_ID*. Finally, the message appended with this information is stored in the structure, which is returned to the application.

4.2 WSSL Receiver

The RD application calls the *WSSL_Receiver*, through the function *init_wssl_rcv*, passing the objects *signed_msg*, *SD_ID*, and the *Struct Receiver*. First, the *signed_msg* passes through the security function, *entity_security_rcv*, where the signature is removed, and if the *SD_ID* and the signature are correct, the *safe_msg* 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 *safe_msg* 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.

4.2.1 *WSSL Receiver Entity for Security.*

WSSL_Receiver's security entity, *entity_security_rcv*, is responsible for removing the signature from the received *signed_msg*. The signature has a fixed 64-byte size, and *fullSize* is obtained from the data appended by the sender to the *signed_msg*. 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 *signed_msg*. At last, the *safe_msg* is retrieved and returned to be used in the *WSSL_Receiver*'s safety entity. The *WSSL_Receiver*'s security general flow is shown in Fig. 3.

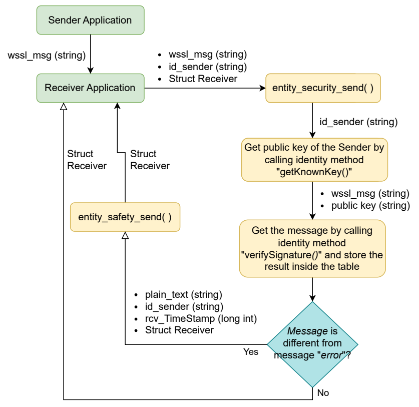


Figure 3: WSSL Receiver's security

4.2.2 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 (illustrated in Fig. 4). These differences are in the check conditions, where each received message is designated with a status related to the sequence number and inter-message delay. The data inside the table is organized in the following format:

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

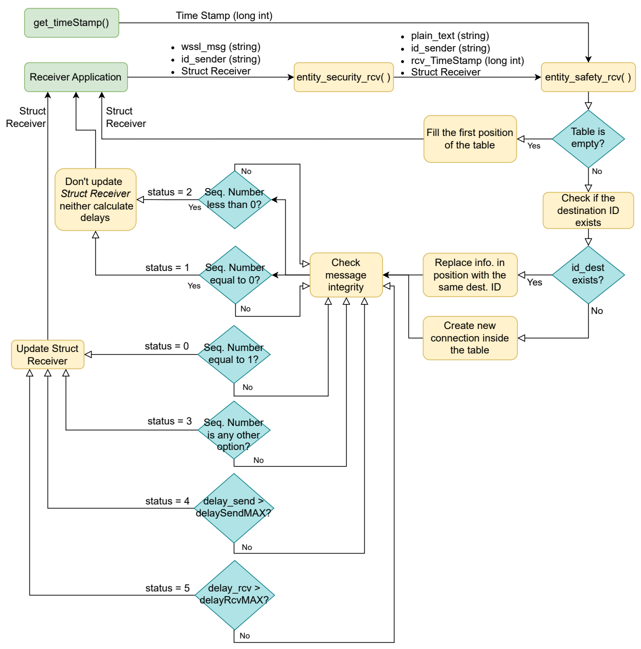


Figure 4: WSSL Receiver's safety

The inter-message delay, *delay_snd*, represents the time delay between the last two received messages. The *delay_snd* 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 task 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.

As it happens with the *delay_snd*, the sequence number is 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. Also, only the last message of each SD_ID is stored inside the table.

If the RD_ID is different 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 sequence number and the last sequence number 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 bigger than 1. Each message status generates a unique table event, which is returned to the $WSSL\_Receiver$ application.

5 EVALUATION

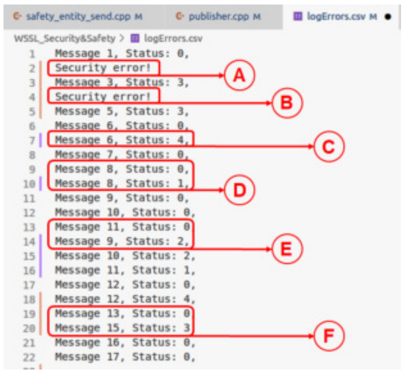
Since this work is developed for Critical Systems, it is important to evaluate the WSSL's efficiency, calculate its implementation costs, and show its ability of detecting errors. For testing purposes, Message Queue Telemetry Transport (MQTT) messaging protocol was chosen to be implemented over the Transport Layer [21], with the Mosquitto Broker. Using MQTT, the publisher must be configured as the $WSSL\_Sender$ and the subscriber as the $WSSL\_Receiver$.

5.1 Generating Errors

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 evaluation, we deliberately sabotaged the message to verify the correct operation of the WSSL.

During the tests, we sent twenty messages with the same SD_ID; within these messages, some samples were chosen to be purposely *sabotaged*. The delay threshold was considered ten milliseconds, and the following six types of security errors were tested (see Fig. 5):

- **(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.



```
Message 1, Status: 0,
Security error!
Message 3, Status: 3,
Security error!
Message 5, Status: 3,
Message 6, Status: 0,
Message 6, Status: 4,
Message 7, Status: 0,
Message 8, Status: 0,
Message 8, Status: 1,
Message 9, Status: 0,
Message 10, Status: 0,
Message 11, Status: 0
Message 9, Status: 2,
Message 10, Status: 2,
Message 11, Status: 1,
Message 12, Status: 0,
Message 12, Status: 4,
Message 13, Status: 0
Message 15, Status: 3
Message 16, Status: 0,
Message 17, Status: 0,
```

Figure 5: Log file of generated errors

Notice that the statuses are WSSL's method for notifying the application about the problem, so handling the error must be the application's responsibility. Also, the statuses related to the delay are analyzed separately by *WSSL_Receiver*. 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.

The analysis of Fig. 5 demonstrates the ability of the WSSL to detect all possible attacks, caused or not by malicious agents, on the receiving device. Furthermore, it has proven capable of detecting and informing the application about network threats arising from these attacks or even network congestion.

5.2 WSSL Network 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, the time to send all messages, the time to receive all messages, and the inter-message delay must be evaluated.

For evaluation tests, a laboratory setup was put together using two desktops with an Intel Pentium CPU G645 2.90 GHz processor and 4 GB of memory RAM running Ubuntu version 20.04.

Ten thousand messages were sent at approximately 333 Hz. This frequency was the lowest frequency possible using the MQTT protocol when aiming for zero lost messages in the RD. Although, even with this limitation, this frequency is sufficient to represent the behavior of hard real-time systems, such as automotive [9] and aerial applications [24]. The *sending time* is considered the total time the *WSSL_Sender* takes to send all messages, and the total *receiving 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).

The tests for both scenarios were repeated ten times for each of the following cases, as presented in Table 1:

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

Table 1: Sending and receiving time when adding WSSL into the communication between devices taking MQTT as a reference.

	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	32160	32160	+223	+223	+3317	+3317	+3456	+3456
T2	32175	32175	+231	+232	+3295	+3295	+3472	+3472
T3	32158	32158	+234	+234	+3309	+3309	+3497	+3497
T4	32170	32170	+231	+231	+3373	+3373	+3488	+3489
T5	32167	32167	+214	+214	+3256	+3256	+3400	+3400
T6	32171	32171	+230	+231	+3319	+3319	+3450	+3450
T7	32179	32179	+218	+218	+3323	+3323	+3423	+3422
T8	32183	32183	+208	+208	+3316	+3317	+3486	+3486
T9	32178	32178	+228	+235	+3337	+3337	+3425	+3425
T10	32167	32167	+217	+217	+3342	+3342	+3466	+3466
Avg	32171	32171	+223	+224	+3319	+3319	+3456	+3456

The WSSL's ability to detect excessive delay was also evaluated. The purpose is to show that the WSSL will alert the application if it exceeds the defined delay threshold. The excessive delay detection accuracy was tested by sending 100 messages at the same frequency and thresholds as before. 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 are plotted in Fig. 6.

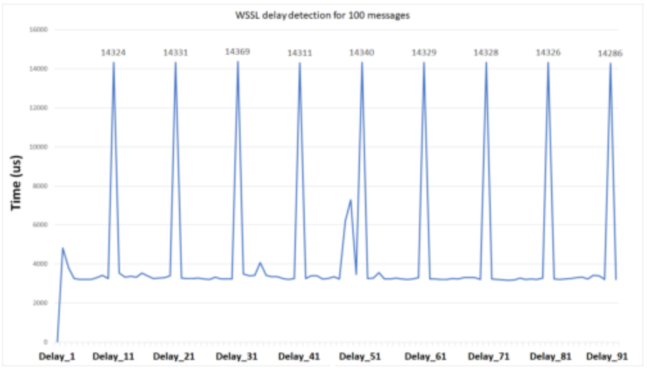


Figure 6: WSSL's delay detection

Implementing the WSSL over a commonly used protocol for IoT applications such as MQTT reinforces its flexible character for other protocols and possible use in other 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.

6 CONCLUSIONS

This paper describes the architecture and the implementation of a Wireless Safety and Security Layer (WSSL) to improve the applicability of 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.

Future research must focus on the enhancement of the safety and security layers. For example, a cryptography method could improve the security layer, increasing the WSSL dependability. In addition, the safety layer methods used for hazard identification may be tested in different environments, not only in simulators but in real applications, for instance, CPS applications implemented in aerial vehicles, cars, residential automation, and others.

ACKNOWLEDGMENTS

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/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.

REFERENCES

[1] Johan Akerberg, Mikael Gidlund, Tomas Lennvall, Jonas Neander, and Mats Björkman. 2011. Efficient integration of secure and safety critical industrial wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking* 2011 (09 2011), 1–13. https://doi.org/10.1186/1687-1499-2011-100

[2] Fredrik Asplund, John McDermid, Robert Oates, and Jonathan Roberts. 2019. Rapid Integration of CPS Security and Safety. *IEEE Embedded Systems Letters* 11, 4 (2019), 111–114. https://doi.org/10.1109/LES.2018.2879631

[3] Hany F. Atlam and Gary B. Wills. 2020. *IoT Security, Privacy, Safety and Ethics* (2nd. ed.). Springer International Publishing, Cham, 123–149. https://doi.org/10.1007/978-3-030-18732-3_8

[4] Ali Balador, Anis Kouba, Dajana Cassioli, Fotis Foukalas, Ricardo Severino, Daria Stepanova, Giovanni Agosta, Jing Xie, Luigi Pomante, Maurizio Mongelli, Pierluigi Pierini, Stig Petersen, and Timo Sukuvaara. 2018. Wireless Communication Technologies for Safe Cooperative Cyber Physical Systems. *Sensors* 18 (11 2018), 4075 pages. https://doi.org/10.3390/s18114075

[5] Márcia C. Rocha and Enio Vasconcelos Filho. 2022. WSSL_Library. https://github.com/marciacr/WSSL_Library

[6] Nelson H. Carreras Guzman, Igor Kozine, and Mary Ann Lundteigen. 2021. An integrated safety and security analysis for cyber-physical harm scenarios. *Safety Science* 144 (12 2021), 105458 pages. https://doi.org/10.1016/j.ssci.2021.105458

[7] Gabriele Cecchetti, Anna Lina Ruscelli, Filippo Cugini, and Piero Castoldi. 2013. An Implementation of EURORADIO Protocol for ERTMS Systems. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering* 7, 6 (2013), 693–702.

[8] European Standards 2010. *EN 50159*. European Standards. https://www.en-standard.eu/ilnas-en-50159-railway-applications-communication-signalling-and-processing-systems-safety-related-communication-in-transmission-systems/

[9] European Telecommunications Standards Institute. 2009. *ETSI TR 102 638 V1.1.1 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions.* Technical Report V1.1.1. European Telecommunications Standards Institute.

[10] Enio Vasconcelos Filho, Nuno Guedes, Bruno Vieira, Miguel Mestre, Ricardo Severino, Bruno Gonçalves, Anis Koubaa, and Eduardo Tovar. 2020. Towards a Cooperative Robotic Platooning Testbed. In *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), 2020.* IEEE, Ponta Delgada, Portugal, 332–337. https://doi.org/10.1109/ICARSC49921.2020.9096132

[11] Enio Vasconcelos Filho, Ricardo Severino, Anis Koubaa, and Eduardo Tovar. 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. FEUP, Porto, Portugal, 3 pages.

[12] Enio Vasconcelos Filho, Ricardo Severino, Joao Rodrigues, Bruno Gonçalves, Anis Koubaa, and Eduardo Tovar. 2021. CopaDrive: An Integrated ROS Cooperative Driving Test and Validation Framework. In *Robot Operating System (ROS)*. Studies in Computational Intelligence, Vol. 962. Springer International Publishing, Cham, 121–174. https://doi.org/10.1007/978-3-030-75472-3_4

[13] Javier Hoffmann, Dirk Kuschnerus, Trevor Jones, and Michael Hubner. 2018. Towards a Safety and Energy Aware protocol for Wireless Communication. In *2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, Lille, France, 1–6. https://doi.org/10.1109/ReCoSoC.2018.8449380

[14] N. Indira, S. Rukmanidevi, and A.V. Kalpana. 2020. Light Weight Proactive Padding Based Crypto Security System in Distributed Cloud Environment:. *International Journal of Computational Intelligence Systems* 13, 1 (2020), 36. https://doi.org/10.2991/ijcis.d.200110.001

[15] Zuzhen Ji, Shuang-Hua Yang, Yi Cao, Yuchen Wang, Chenchen Zhou, Liang Yue, and Yinqiao Zhang. 2021. Harmonizing safety and security risk analysis and prevention in cyber-physical systems. *Process Safety and Environmental Protection* 148 (04 2021), 156–178. https://doi.org/10.1016/j.psep.2021.03.004

[16] Li jie Chen, Zhen yu Shan, Tao Tang, and Hong jie Liu. 2011. Performance analysis and verification of safety communication protocol in train control system. *Computer Standards & Interfaces* 33, 5 (2011), 505–518. https://doi.org/10.1016/j.csi.2011.02.006

[17] Georgios Kavallieratos, Sokratis Katsikas, and Vasileios Gkioulos. 2020. Cybersecurity and Safety Co-Engineering of Cyberphysical Systems—A Comprehensive Survey. *Future Internet* 12 (04 2020), 505–518. https://doi.org/10.3390/fi12040065

[18] Kai Li, Wei Ni, Yousef Emami, Yiran Shen, Ricardo Severino, David Pereira, and Eduardo Tovar. 2020. Design and Implementation of Secret Key Agreement for Platoon-based Vehicular Cyber-physical Systems. *ACM Transactions on Cyber-Physical Systems* 4, 2 (04 2020), 1–20. https://doi.org/10.1145/3365996

[19] Micaela Caserza Magro, 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)*. IEEE, Florence, Italy. https://doi.org/10.1109/EEEIC.2016.7555402

[20] Christoph Schmittner, Zhendong Ma, Carolina Reyes, Oliver Dillinger, and Peter Puschner. 2016. Using SAE J3061 for Automotive Security Requirement Engineering. In *Computer Safety, Reliability, and Security*. Lecture Notes in Computer Science, Vol. 9923. Springer International Publishing, Cham, 157–170. https://doi.org/10.1007/978-3-319-45480-1_13

[21] Diego R. C. Silva, Guilherme M. B. Oliveira, Ivanovitch Silva, Paolo Ferrari, and Emiliano Sisinni. 2018. Latency evaluation for MQTT and WebSocket Protocols: an Industry 4.0 perspective. In *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Natal, 01233–01238. https://doi.org/10.1109/ISCC.2018.8538692

[22] The 61508 Association 2010. *IEC 61508* (2nd ed.). The 61508 Association. https://www.61508.org/index.php

[23] Alexey Vinel, Nikita Lyamin, and Pavel Isachenkov. 2018. Modeling of V2V Communications for C-ITS Safety Applications: A CPS Perspective. *IEEE Communications Letters* 22, 8 (2018), 1600–1603. https://doi.org/10.1109/LCOMM.2018.2835484

[24] Melih Yildiz, Burcu Bilgiç, Utku Kale, and Dániel Rohács. 2021. Experimental Investigation of Communication Performance of Drones Used for Autonomous Car Track Tests. *Sustainability* 13, 10 (May 2021), 5602. https://doi.org/10.3390/su13105602

[25] Xiang-Yu Zhou, Zheng-Jiang Liu, Feng-Wu Wang, and Zhao-Lin Wu. 2021. A system-theoretic approach to safety and security co-analysis of autonomous ships. *Ocean Engineering* 222 (02 2021), 108569 pages. https://doi.org/10.1016/j.oceaneng.2021.108569