Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Challenges in the Industrial Internet of Things Scenario: wireless systems for functional safety, distributed measurement and real-time communication

Ph.D. candidate
Alberto Morato

Advisor
prof. Stefano Vitturi

Director & Coordinator
prof. Andrea Neviani

Ph.D. School in
Information Engineering

Department of
Information Engineering
University of Padova
2021

Challenges in the Industrial Internet of Things Scenario: wireless systems for functional safety, distributed measurement and real-time communication

Ph.D. candidate: Alberto Morato

Director: prof. Andrea Neviani
Advisor: prof. Stefano Vitturi

Ph.D. School in Information Engineering
Department of Information Engineering
2021

"The problem is you think you have time. The hours have a way of turning into days, the days into weeks, the weeks into years, and the years into a lifetime. How different would your life and memories be today (or ten years later) if you had done that?"

Shawn James

Abstract

The changes introduced by Industry 4.0 are profoundly revolutionizing the way production systems are conceived in many sectors. In the past, production units in industrial plants were conceived as monolithic entities in which elements communicated and cooperated only within the unit itself. The insertion of new elements sometimes required a complete redesign and configuration of the communication network. Today, in a very similar way to what is happening in the consumer world with the Internet of Things, the Industrial Internet of Things (IIoT) is progressively taking hold in the industrial scenario. This paradigm enables all the components of a production unit to be part of a global and distributed network of smart devices that interact in order to improve the production process. The effectiveness of this smart environment is based on pervasive and ubiquitous connectivity, often achieved via wireless systems, able to ensure a fast, safe, reliable, and contextualized transfer of information. All these characteristics play a fundamental role in IIoT domains where coordination between sensors, actuators, and control systems is required, for example, on those implements human–machine interaction. In this thesis, we address such a topic, focusing on the challenges in integrating real-time distributed measurement systems and safety critical systems in IIoT scenarios. Innovative solutions in this context will be proposed and verified through extensive testing and simulations.

Wireless–based Functional safety networks – The first part of this thesis focuses on the use of safety networks and protocols in an IIoT ecosystem. In particular, we will discuss the issues and challenges of adopting safety protocols in wireless communication networks by analyzing the results obtained from a test campaign conducted on a prototype of such a network. Moreover, the actual applicability in an industrial context will be verified and discussed through a simulated environment.

Industrial IoT-Based Distributed Measurement Applications – The second part addresses distributed measurement systems in the IIoT domain. First, the possible use of protocols designed for Machine–to–Machine (M2M) communications to exchange information from the field level up to the cloud between distributed sensors is analyzed. Next, test results conducted in some implementations of this protocol will be reported and discussed focusing on power consumption, computational complexity, and communication time. In order to provide a meaningful evaluation, the tests were performed in hardware platforms oriented to embedded devices using both wired and wireless communication systems.

Sommario

I cambiamenti introdotti da Industria 4.0 stanno rivoluzionando profondamente il modo di concepire i sistemi produttivi in molti settori. In passato, le unità produttive negli stabilimenti industriali erano concepite come entità monolitiche in cui gli elementi comunicavano e cooperavano solo all'interno dell'unità stessa. L'inserimento di nuovi elementi talvolta richiede una completa riprogettazione e configurazione della rete di comunicazione. Oggi, in modo molto simile a quanto sta accadendo nel mondo consumer con l'Internet of Things, l'Industrial Internet of Things (IIoT) sta progressivamente prendendo piede nello scenario industriale. Questo paradigma consente a tutti i componenti di un'unità produttiva di essere parte di una rete globale e distribuita di dispositivi intelligenti che interagiscono per migliorare il processo produttivo. L'efficacia di questo ambiente intelligente si basa su una connettività pervasiva e ubiqua, spesso ottenuta tramite sistemi wireless, in grado di garantire un trasferimento di informazioni veloce, sicuro, affidabile e contestualizzato. Tutte queste caratteristiche giocano un ruolo fondamentale nei domini IIoT dove è richiesto il coordinamento tra sensori, attuatori e sistemi di controllo, ad esempio su quelli che implementano l'interazione uomo–macchina.

In questa tesi, affrontiamo tale argomento, concentrandoci sulle sfide nell'integrazione di sistemi di misura distribuiti in tempo reale e sistemi critici per la sicurezza negli scenari IIoT. Soluzioni innovative in questo contesto verranno proposte e verificate attraverso approfonditi test e simulazioni.

Functional safety tramite reti wireless – La prima parte della tesi è incentrata sull'uso di reti e protocolli safety in un ecosistema IIoT. In particolare, verranno discusse le problematiche e le sfide dell'adozione di protocolli safety in reti di comunicazione wireless analizzando i risultati ottenuti da una campagna di test condotta su un prototipo di tale rete. Verrà inoltre verificata e discussa l'effettiva applicabilità in un contesto industriale attraverso un ambiente simulato.

Industrial IoT-Based Distributed Measurement Applications – La seconda parte della tesi affronta il problema dei sistemi di misura distribuiti in ambito IIoT. In primo luogo verrà analizzato il possibile utilizzo di protocolli ideati per le comunicazioni Machine–to–Machine (M2M) per l'interscambio di informazioni dal livello di campo fino al cloud tra sensoristica distribuita. In seguito, verranno riportati e discussi risultati di test condotti in alcune implementazioni di questo protocollo con particolare attenzione al consumo energetico, complessità computazionale e tempi di comunicazione. Al fine di fornire una valutazione significativa, i test sono stati effettuati in piattaforme hardware orientate ai dispositivi embedded utilizzando sistemi di comunicazione sia cablati che wireless.

Ringraziamenti

L'ambiente, le situazioni ma soprattutto le persone che ci circondano durante momenti importanti della vita sono quelle che definiscono chi siamo. Il raggiungimento di questo obiettivo è stato possibile grazie a tutte le persone che in questi anni mi hanno accompagnato lungo questo percorso e che con il loro supporto hanno contribuito a rendermi ciò che sono oggi. A tutti voi va il mio più sentito ringraziamento.

Grazie al mio PhD advisor Prof. Stefano Vitturi, che fin dall'inizio di questo percorso, è sempre stato presente con prezioso ed inesauribile supporto. Un mentore che mi ha fatto crescere come ricercatore, che mi ha aiutato a plasmare ogni aspetto della mia carriera e che mi ha sempre consigliato per il meglio mettendo le mie ambizioni e la mia felicità sempre al primo posto.

Grazie anche a Prof Angelo Cenedese e Prof Federico Tramarin per gli innumerevoli consigli e tutto il lavoro fatto insieme.

Un ringraziamento va al mio tutor aziendale Giampaolo Fadel, ai colleghi dell'ufficio software e a tutta la CMZ Sistemi Elettronici. In questi anni in azienda ho trovato un ambiente stimolante, formato da persone con un inestimabile valore tecnico ed umano. L'avermi coinvolto in progetti ambizioni, avermi permesso di condividere e scambiare idee, per me è stato una costate fonte di arricchimento tecnico e stimolo alla ricerca dell'innovazione.

Un ringraziamento va a Christian Rodondi, amico di infanzia ma per me come un fratello. A volte passano lunghi periodi, ma quando ci ritroviamo è come se non fosse passato un giorno. Ripensare ai bei momenti di infanzia passati a giocare insieme e riviverli ancora oggi con le lunghe chiacchierate e le escursioni in MTB, mi ha davo molte volte la forza di andare avanti. Ti sono grato per i bei momenti passati insieme.

Mi rattrista il non poter condividere questo importante traguardo con due persone care che non ci sono più. Caro papà, grazie per avermi insegnato a perseguire con tenacia i miei obiettivi. Nel corso degli anni non sono mancati momenti in cui persone, o addirittura io stesso, pensassero che non sarei riuscito a raggiungere quelli che per me sono stati obiettivi importanti. Però, in qui momenti, ho sempre tenuto a mente le tue parole: *volere è potere*. Il ricordo di queste parole e dei tuoi insegnamenti sono ciò che mi ha aiutato ad essere ciò che sono. Te ne sono riconoscente di tutto cuore. Grazie anche a te zio, che fin da quando ero piccolo hai alimentato e supportato la mia passione per la scienza e la tecnologia.

Nel momento in cui scrivo questi ringraziamenti, ho da pochi giorni dovuto salutare per l'ultima volta la mia cagnolina Lucy. Quella che abbiamo condiviso è stata una piccola parentesi di vita. Eri una piccola luce che portava gioia nei momenti più tristi e di solitudine. Ricorderò con affetto e malinconia le camminate nei nostri amati boschi dell'altopiano.

Infine, il ringraziamento più importante va a mia madre. Graziella, ti sono grato per i sacrifici e per avermi continuamente supportato e incoraggiato in tutte le scelte, dentro e fuori la carriera universitaria. Tutti gli obiettivi che ho raggiunto e raggiungerò in futuro sono merito anche del tuo infallibile sostegno. A te la mia più sconfinata gratitudine.

Alberto Morato
Padova, 08/09/2021

Contents

List of Figures

List of Tables

Acronyms

ACK Acknowledge 15

AGV Autonomous Guided Vehicle 7

AI Artificial Intelligence 1

API Application Programming Interface 76, 77

CAGR Compound Annual Growth Rate 8

CAN Controller Area Network 15

COB Communication Object 15

Cobot Collaborative Robot 7

COM Component Object Model 76, 77

CPF Communication Profile Family 11

CPU Central Processing Unit 75, 79, 83, 85, 86, 87, 88, 89, 94, 95, 96, 97, 98, 99, 100, 105

CRC Cyclic Redundancy Check 15, 20, 21, 23, 27, 29, 39

CTS Clear To Send 33

DCOM Distributed Component Object Model 76, 77

DMS Distributed Measurement System 73

DVFS Dynamic Voltage and Frequency Scaling 97, 98, 100

EPDF Empirical Probability Density Function xiii, xiv, 31, 32, 35, 43, 44, 45, 70, 89, 94, 95, 96

FSF Functional Safety Fieldbus 7, 8, 10, 17

FSoE Fail Safe over EtherCAT xiii, xvii, 4, 5, 17, 20, 21, 22, 23, 26, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40, 43, 45, 46, 47, 49, 50, 51, 52, 57, 59, 62, 64, 67, 104, 105

GPIO General Purpose Input/Output 95, 96

HMI Human Machine Interface 76

HTTP Hypertext Transfer Protocol 77

IIoT Industrial Internet of Things xiii, 1, 7, 8, 26, 73, 74, 75, 80, 82, 85, 94, 100, 103, 104

IoT Internet of Things 1, 3

IP Internet Protocol 27, 28, 88

IT Information Technology 1

MAC Medium Access Control 26

MIMO Multiple Input Multiple Output 59, 62

MQTT Message Queue Telemetry Transport 1, 59

MSE Mean Square Error 53, 55

NCS Networked Control System 59

OLE Object Linking and Embedding 76

OMNeT++ Objective Modular Network Testbed in C++ xiv, xvii, 5, 49, 50, 51, 65, 66, 68, 69, 104

OOP Object Oriented Programming 78

OPC Open Platform Communications 76, 77

OPC-UA Open Platform Communications-Unified Architecture xiii, xiv, xvii, 1, 5, 8, 17, 18, 19, 20, 74, 75, 76, 77, 78, 79, 80, 82, 83, 84, 85, 86, 87, 88, 89, 91, 94, 95, 96, 98, 99, 100, 105

PC Personal Computer 77

PDU Protocol Data Unit 21, 23, 37, 46, 104

PER Packet Error Rate xiii, xvii, 42, 43, 47, 50, 57, 66

PLR Packet Loss Rate xiii, xvii, 42, 43, 47

QoS Quality of Service 26, 37

REP Residual Error Probability 13

RF Radio Frequency 52

ROS Robot Operating System 26

RTE Real-Time Ethernet 7, 8, 18, 59, 103

RTS Request To Send 33

SCADA Supervisory Control And Data Acquisition 76

SCT Safeguard Life Time xiii, 15, 16

SFRT Safety Function Response Time 5, 26, 47, 59, 60, 61, 62, 63, 64, 65, 66, 69, 70, 71, 105

SIL Safety Integrity Level 7, 10, 12, 13, 25, 46

SIMO Single Input Multiple Output 59

SISO Single Input Single Output 59, 60

SNR Signal to Noise Ratio 50, 57, 69

SOAP Simple Object Access Protocol 77

SoC System on Chip 27

SPDU Safety Protocol Data Unit 18, 19, 20, 22, 34, 37, 38, 39, 42, 43, 45, 47, 52, 64, 66

SRDO Safety Relevant Data Object xiii, 15

SRVT Safety Relevant Validation Time xiii, 15

STO Safe Torque Off 67, 68

TCP	Transmission Control Protocol 4, 26, 27, 37, 42, 43, 44, 45, 46, 47, 77, 88, 104
TDMA	Time Division Multiple Access 26
TLS	Transport Layer Security 77
TSN	Time-Sensitive Networking 1, 8, 18, 26, 47, 80
UDP	User Datagram Protocol 4, 26, 27, 28, 37, 38, 39, 42, 43, 44, 45, 46, 47, 49, 52, 66, 80, 96, 104
UDPc	UDP with caching layer 4, 42, 43, 44, 45, 46, 47
WLAN	Wireless Local Area Network 26, 27, 34, 59
WNIC	Wireless Network Interface Controller 52
WSN	Wireless Sensor Network 25, 26
XML	eXtensible Markup Language 77

1
Introduction

1.1 Industry 4.0

The third industrial revolution began in the early '70s, has completely transformed several economic sectors by progressively introducing instruments typical of Information Technology (IT) in the industrial field. Undoubtedly, one of the most important innovations was the progressive digitalization of industrial controls and automation systems, thanks to communication networks. This wave of digitalization is still so strong today that we are witnessing the fourth industrial revolution. This technological push, also defined as Industry 4.0, has the Internet of Things, Cyber–Physical Systems, Big Data Analytics, Cloud Computing, Autonomous Robotics, Augmented Reality, and Additive Manufacturing as its fundamental cornerstones. In this new and vast scenario, the boundaries between devices, production processes, and services are gradually disappearing, giving way, as defined by van Kranenburg and Dodson (2008), to *"a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "Things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network."* Some fundamental aspects of the IIoT landscape emerge from this definition. First of all, is the pervasive connection between fixed and mobile devices, which necessarily will often require the massive use of ultra-low latency, high throughput, ultra-reliable, secure, and fail-safe hybrid wired-wireless connections (Li, Xu, and Zhao (2018b)). According to Wijethilaka and Liyanage (2021), it is expected that the number of interconnected Internet of Things (IoT) devices will be 27 billion by 2024. Real-time wired networks, Wi-Fi 6, and 5G enabled IoT, will connect a massive number of smart devices and contribute to meeting market demand for highly integrated services to stimulate new economics and social development (Adame, Carrascosa, and Bellalta (2020a); Dawy, Saad, Ghosh, Andrews, and Yaacoub (2015); Li et al. (2018b); Maldonado, Karstensen, Pocovi, Esswie, Rosa, Alanen, Kasslin, and Kolding (2021); Sudhakaran, Montgomery, Kashef, Cavalcanti, and Candell; Wijethilaka and Liyanage (2021)). A second fundamental aspect is the contextualization of exchanged information. Given the large number of heterogeneous devices that share the same global communication network and that must be able to self-configure and communicate with each other, it is of paramount importance to understand the context of collected data to perform effective processing and decision making (Yavari, Jayaraman, Georgakopoulos, and Nepal (2017)). In other words, the IT infrastructure supporting these smart systems must be able to both aggregate and categorize information to be better managed and used. For example, modern predictive maintenance systems base on Artificial Intelligence (AI), need to aggregate and categorize massive amount of data from countless distributed measurement devices and sensors (U, Yang, Cai, Karlapalem, Liu, and Huang (2020)). On the other hand, is necessary to filter and exclude data irrelevant to the specific application, improving processing speed and information extraction in embedded IoT applications (Maniglia and Sofia (2019)). In this regard, examples are MQTT and OPC-UA pub–sub which, thanks to their publisher–subscriber structure, can achieve those tasks seamlessly. Another crucial aspect of such networks is the real-time behavior. Especially in wireless networks, a common pitfall is the high intrinsic sensitivity to channel condition variations due, for example, to external radio interference. This often leads to consequences such as high transmission failure probability and the introduction of delays and jitter that can compromise the system's real-time behavior. In IIoT systems, these problems can appear even more evident. Due to a large amount of data to be exchanged and the bandwidth that must be shared between many devices, the probability of possible collisions with a consequent increase in latency can be even more marked. Not only that but the lack of timeliness can also be introduced by the applications that make use of this IIoT ecosystem. For example, the use of non-real-time operating systems, computationally heavy application layer protocols, or the use of low-power embedded devices certainly have a negative impact on the real-time performance of the system. In distributed measurements systems and safety-critical systems, this is critically important since they are often required to provide very tight timing performance. Both require fast and reliable data transmission, which, if not guaranteed, can lead to catastrophic consequences. Finally, there is a need to have standardized communication systems. In the past years, especially in the industrial field, there has been a fragmentation of technologies for data transmission. Many manufacturers have proposed proprietary protocols, often with similar but completely non-interoperable functionality. In recent years, thanks to the standardization requirements introduced by Industry 4.0, the trend has been reversed, and there is a progressive unification of the communication systems that make it easier to interact between devices from different manufacturers. This revolution is not only affecting protocols but also the way of conceiving devices. For example, the recent introduction of Time-Sensitive Networking (TSN) standards in both the industrial and consumer sectors makes it possible to coexist both time–sensitive and best-effort applications on the same network, probably making dedicated real-time networks unnecessary. This suggests that in the future, there will no longer be a distinction between IoT and IIoT, but we will come to have an Internet of Everything (Chatzigiannakis and Tselios (2021)).

While access to information is made easier thanks to this global network of things, this hyperconnectivity entails cyber threats with significant risks for society, including business, environment, and health. IoT safety and security is one of the highest priorities to prevent these systems from causing or inducing unwanted physical damage and threats to the surrounding environment. Generally speaking, the security problem in IoT systems is of fundamental importance because, if hacked, devices' functionality can be manipulated, causing damage to the controlled objects and endangering people in contact with it. The same considerations are also valid for safety. Today's systems are so complex that usually it is difficult to predict and test all the working conditions. Therefore, it is mandatory that in case of faults, communication problems, or unexpected behavior, these devices are fail-safe. Ensuring safety and reliability in IoT systems, undoubtedly, is not an easy task as it entails high hardware complexity, particular network protocols, timeliness, and predictability in systems whose functionality, configuration, or topology may change quickly. However, it is ethically binding to continue to explore the risks and opportunities associated with IoT-driven systems, developing new design approaches to protect the privacy and safety of users.

Looking specifically at the industrial environment, traditional static systems, interconnected by hard-wired links or wired networks, have the advantage of having almost always predictable behaviors. However, with the progressive transformation into smart factories that are taking place in recent years, the use of autonomous and reconfigurable systems is increasing. For example, semi-autonomous and fully autonomous robots have been used to perform physically intensive and dangerous tasks in several industries. Other examples are collaborative robots intended to work side-by-side with humans, possibly entailing physical interactions. Several challenges arise analyzing these emerging technologies from a functional safety prospective. Adjectives such as distributed, autonomous, and self-configuring imply the presence in the system of a fair amount of "artificial intelligence" capable of adapting individual systems to different working conditions. In general, these characteristics lead the communication system and infrastructures to evolve dynamically during operation, making it difficult, if not impossible, to predict the entire structure during design and development. All these factors lead to uncertainties or properties that are difficult to predict and that, beyond certain limits, can contradict or make the concept of functional safety inapplicable. For this reason, safety could quickly become a bottleneck in the transition to Industry 4.0 because, despite its high economic potential, innovation should never come at the expense of safety. Therefore, in this context, current, and future research should focus on analyzing ideas, solutions, tools, and technologies that can make this transition possible without significant impediments.

1.2 Contribution

This manuscript addresses the issues discussed so far by analyzing the challenges and providing innovative solutions concerned with functional safety networks and distributed measurements systems in real-time Industrial Internet of Things ecosystems.

In *Chapter* 2, we first introduce the design characteristics of functional safety networks and protocols. These are currently adopted in combination with wired real–time communication networks in environments where the transmission of safety–relevant messages is required to implement distributed functional safety systems integrated with the factory communication infrastructure. Two main protocols will be analyzed, namely those based on a white channel and black channel approach. In particular, the latter, thanks to particular design characteristics, can potentially be used on means of communication other than those for which they were originally conceived. Finally, particular insight on the design characteristics of the Fail Safe over EtherCAT (FSoE) will be provided. Part of the contributions reported in *Chapter* 2 is based on Peserico, Morato, Tramarin, and Vitturi (2021c).

The implementation of the FSoE protocol on top of IEEE802.11 is considered in *Chapter* 3. The main idea supporting our implementation is based on the fact that protocols on top of the black channel are not aware of the transport protocol nor the transmission medium used to carry the safety data. Hence, the black channel approach represents an ideal approach for porting functional safety protocols over those wireless industrial communication systems not having a specific safety extension. After a detailed overview of the implementation, we discuss the extensive tests carried out on a prototype implementation focusing on the polling time, i.e., the time needed to complete an exchange of safety data and the packet loss rate, reflecting the maximum achievable safety level. The contributions reported in *Chapter* 3 are based on Morato, Vitturi, Cenedese, Fadel, and Tramarin (2019).

The prototype implementation of FSoE on top of IEEE802.11 revealed some concerns about the packet loss, which makes the implementation unsuitable for the most demanding safety system. To overcome the problem, in *Chapter* 4, we assessed different transport layer protocols, in particular UDP and TCP. However, even if TCP seems the best candidate thanks to flow control and quality of service mechanisms, it lacks multicast and broadcast messaging useful in cases where faster addressing of nodes in the network is required. For this reason, we have proposed a protocol-agnostic caching layer that can bring support for frame retransmission and duplicate message management in any layer of the protocol stack. The algorithm and the working principle of this layer have been discussed in detail. In addition, the three variants, namely TCP, UDP, and UDPc, have been compared with each other both through a simulated environment and on a prototype, analyzing, in particular, the impact on polling time and packet loss.

In *Chapter* 5, we introduce a simulation model for functional safety over wireless based on OMNeT++. The main purpose of the simulator is to be able to analyze safety-critical distributed systems with multiple nodes on different scenarios and environmental conditions. The tuning of the model has been carried out based on measurements performed on an experimental setup specifically designed for this purpose. In particular, the channel error model and the polling time have been tuned for different conditions of the communication channel. The simulator has been partially tested and validated considering an industrial scenario comprising multiple nodes. The contributions reported in *Chapter* 5 are based on Morato, Peserico, Fedullo, Tramarin, and Vitturi and Peserico, Fedullo, Morato, Tramarin, and Vitturi (2021b).

In *Chapter* 6, we address the implementation of a framework to estimate the Safety Function Response Time (SFRT), an important metric to characterize the responsiveness of distributed safety systems. The framework is based on the definition of SFRT given by IEC 61784-3-3, extended by Pimentel and Nickerson (2014), and further modified to fit the specific use case with FSoE. The proposed framework has been partially validated comparing theoretical values of the SFRT calculated used worst–case scenarios polling time obtained in *Chapter* 5, with the actual SFRT obtained with simulation on the calibrated model. The SFRT has also been estimated considering a real industrial use case scenario comprising multiple autonomous mobile robots. Part of the contributions reported in *Chapter* 6 is based on Peserico et al. (2021c).

Finally, *Chapter* 7 deals with distributed measurement systems. As stated before, the large-scale adoption and development of Industry 4.0 require standardized communication protocols capable of operating in different contexts where interaction between heterogeneous devices is required. A newly developed protocol capable of satisfying these requirements is OPC-UA. It is an object-oriented protocol, i.e., the information transmitted is contextualized thanks to attributes associated with the information itself. These characteristics certainly increase the degree of usability of the information transmitted at the expense of complexity that may have a negative impact on the performance, especially in low power embedded devices. In this chapter, we assess different OPC-UA solutions deployed in distributed measurements applications. Some metrics, such as CPU usage, communication time on Ethernet and Wi–Fi, and power consumption, were evaluated on an experimental setup comprising low–power embedded devices. The contributions reported in *Chapter* 7 are based on Morato, Vitturi, Tramarin, and Cenedese (2020) and Morato, Vitturi, Tramarin, and Cenedese (2021).

2 Communication Networks for Industrial Functional Safety

At the beginning of the '90s, during the third industrial revolution, Real-Time Ethernet (RTE) communication networks in industrial automation have been progressively enforced. Indeed, thanks to the improvements brought by such networks in terms of timeliness, reliability, and scalability opened up the possibility of connecting industrial devices deployed over large distributed plant areas (Vitturi, Zunino, and Sauter (2019)). At the same time, the functional safety systems of industrial plants have also begun to evolve. They have also gradually moved from hard-wired solutions to communication networks. One of the fundamental aspects is that these new solutions are able to guarantee the same degree of safety as the hard-wired solutions.

With the advent of Industry 4.0 and the Industrial Internet of Things (IIoT), the situation is further evolving. As represented in Figure 2.1, this new paradigm allows the interconnection of cloud systems, controllers, industrial equipment, sensors/actuators, and any other object within an automation system to each other. The pervasive communication infrastructures, possibly wireless, enable all the devices to be part of wide and integrated factory networks and measurement systems, where operators cooperate with machinery becoming part of the production process itself. Thus, wireless industrial communication networks are becoming essential in order to introduce a completely new category of mobile and possibly autonomous manufacturing equipment, such as Collaborative Robots (Cobots) and Autonomous Guided Vehicles (AGVs) (Robinson (2019)). Nevertheless, applying safety concepts to Industry 4.0 brings several challenges. Indeed, as pointed out by Liggesmeyer and Trapp (2017), the central element of Industry 4.0 defined as "network of autonomous, situational self-controlling, self-configuring, knowledge-based, sensor-supported, and spatially distributed production resources", introduce uncertainty in the production system generally in conflict with safety requirements which usually expects deterministic and predictable behaviors. Moreover, IIoT is increasingly introducing cybersafety and cybersecurity issues into the physical world, creating a vast number of new vulnerabilities that can undermine the safety of people and industrial plants (Munirathinam (2020)). Moving from wired to wireless communication has hence become a hot and challenging research topic, especially concerning functional safety. Indeed, as pointed out by Islam, Shen, and Wang (2012), it is necessary that in this new context, wireless communications must be able to ensure the same performance in terms of Safety Integrity Level (SIL) as their wired Functional Safety Fieldbus (FSF) counterpart, despite their error–prone nature (Meany (2017)).

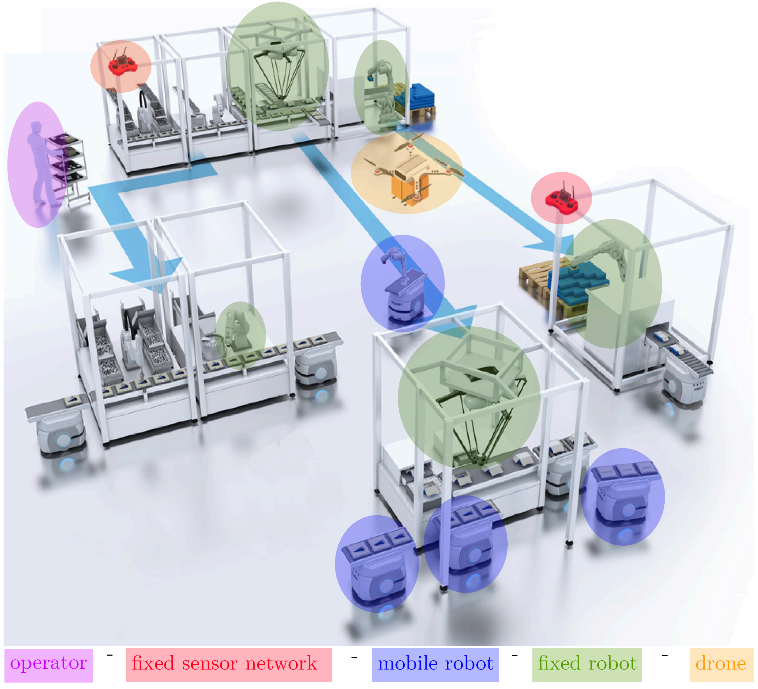

operator - fixed sensor network - mobile robot - fixed robot - drone

Figure 2.1: IIoT warehouse. Inspired by Omron

In this perspective, new emerging technologies such as Time-Sensitive Networking (TSN) and Open Platform Communications-Unified Architecture (OPC-UA) safety have been proposed in Etz, Fruhwirth, and Kastner (2020) to design self-organizing safety systems able to automatically generate a suitable safety configuration based on the working conditions. Combining these two technologies allows obtaining real-time and vendor agnostic communication networks capable of interconnecting devices from the field level to the cloud. Furthermore, due to their design nature, OPC-UA and its safety extension can operate natively on wireless (Morato et al. (2021)) and mobile networks (Abukwaik, Gogolev, Groß, and Aleksy (2020); Kim, Jo, and Jeong (2019)). However, despite the increasing adoption of these new technologies proposed by Industry 4.0, several plants still use the traditional, perhaps legacy, Fieldbus or RTE protocols and their safety extensions. Previous studies in this area of research, like the one conducted by Etz, Fruhwirth, Ismail, and Kastner (2018) and confirmed by HMS Networks (2020), there are five dominant technologies in the industrial Ethernet market segment, none of which are part of the recent innovations introduced by Industry 4.0. As shown in Figure 2.2, EtherNet/IP, PROFINET, EtherCAT, Modbus-TCP, and POWERLINK shares about 50% of the real-time communication networks market with an annual growth of 5%. The interest in these communication protocols is growing as well as their safety extensions. Indeed, as Mordor Intelligence (2021) reported, the functional safety global market, which includes several device types such as safety sensors/actuators, PLCs, is expected to grow with a Compound Annual Growth Rate (CAGR) greater than 8% in the period 2021–2026. One major downside of these networks and their safety extension is that none of those are natively conceived to work with wireless communication channels. However, thanks to their design, they can theoretically be modified and used in the latter. In this chapter, some possibilities in this regard are explored. In particular, after a description of the fundamental characteristics of FSFs, an implementation of one FSF on the IEEE 802.11 is presented, and the results of an extensive experimental session carried out on such a network will be discussed.

This chapter is mainly based on the works in Morato et al. (2019); Vitturi, Morato, Cenedese, Fadel, Tramarin, and Fantinel (2018); Peserico et al. (2021c); Morato et al. and Peserico et al. (2021b).

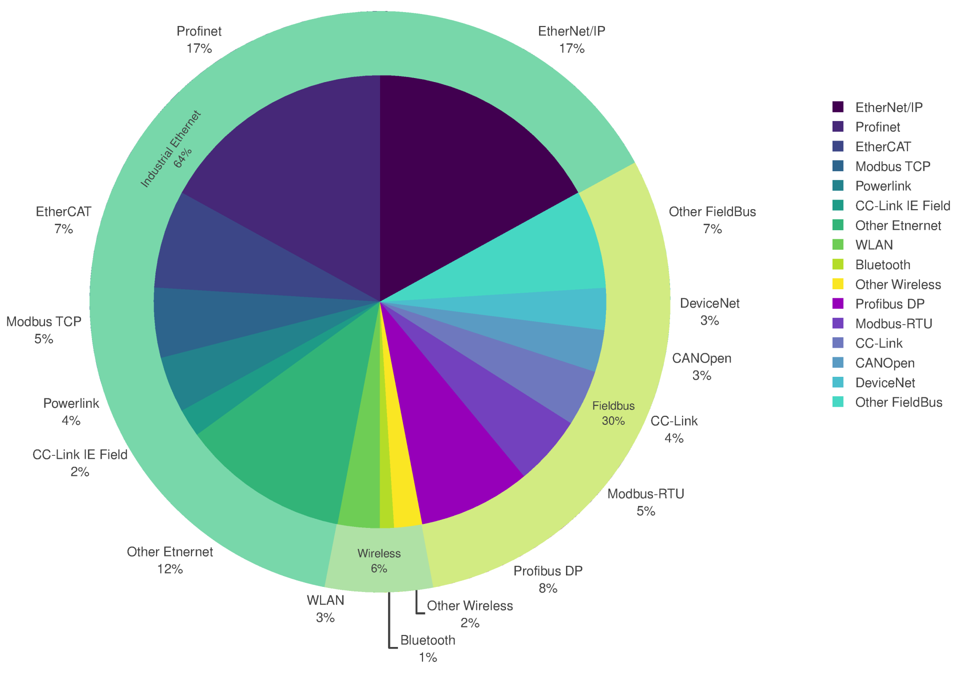


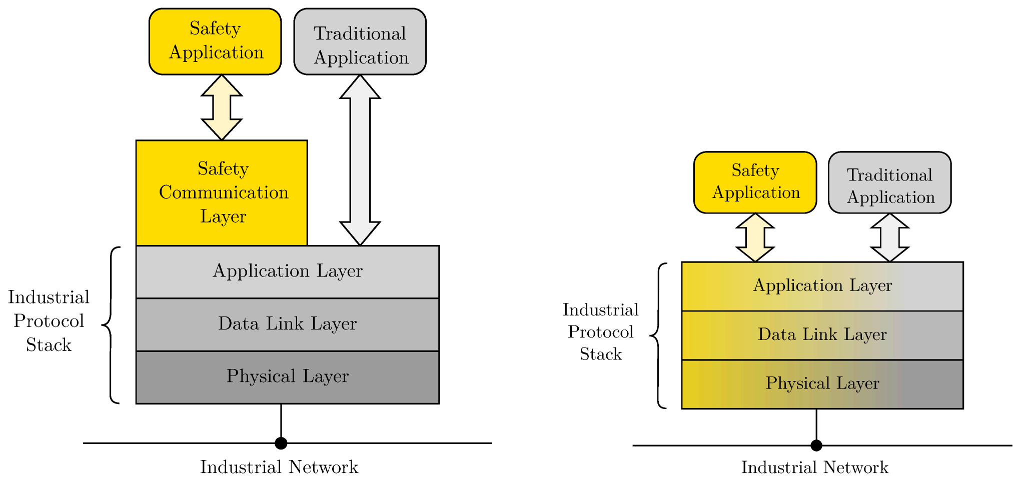
Figure 2.2: Industrial network market shares 2020 according to HMS Networks

2.1 Functional Safety Fieldbuses

Functional Safety Fieldbuses (FSFs), i.e., networks used to transmit safety–related messages as defined by the IEC 61784-3 international standard, play an essential role in ensuring functional safety in production systems. The standard uses the term "Fieldbuses", although the protocols it introduces are (or can be) adopted by any industrial networks, e.g., Real–Time Ethernet networks. We will maintain such a term throughout the manuscript for compliance with the standard.

FSFs are networks designed to connect safety nodes to implement distributed functional safety systems integrated with the factory communication infrastructure that replace and improve the traditional (less performing) safety systems based on dedicated hardware circuits. Indeed, FSFs allow the transmission of safety–related information among nodes with very low error probability to cope with the strong requirements imposed by the IEC 61508 Functional Safety International Standard. For example, to achieve a Safety Integrity Level (SIL) 3, an FSF has to ensure a residual error rate less than $10^{-9}/h$, which represents a tighter performance than that typically provided by traditional fieldbuses or real–time Ethernet networks.

From the protocol architecture point of view, the nodes of an FSF are characterized by a Safety Communication Layer, placed on top or part of their protocol stacks, as described in Figure 2.3.



(a) Black channel **(b)** White channel

Figure 2.3: Protocol Stack of a Functional Safety Node

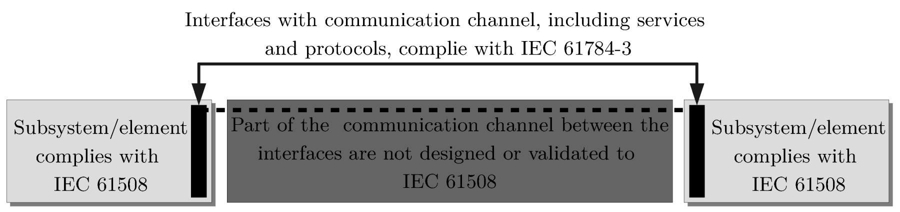
IEC 61784-3 is related to the Communication Profile Families (CPFs) introduced in the fieldbus standardization framework, which started with IEC 61158-1. In this context, IEC 61784-3 defines the functional safety protocols implemented by the safety communication layers. Table 2.1 lists the CPFs for which functional safety protocols have been defined along with their commercial names.

CFP	Commercial Name	Functional Safety Protocol
1	FOUNDATION Fieldbus	FF–SIS
2	Common Ind. Protocol (CIP)	CIP Safety
3	PROFIBUS & PROFINET	PROFIsafe
6	INTERBUS	INTERBUS Safety
8	CC–Link	CC–Link Safety
12	EtherCAT	Fail Safe over EtherCAT (FSoE)
13	Ethernet POWERLINK	Ethernet POWERLINK Safety
14	EPA	EPASafety
16	SERCOS	CIP Safety
17	RAPIEnet	RAPIEnet Safety
18	SafetyNET p	SafetyNET p

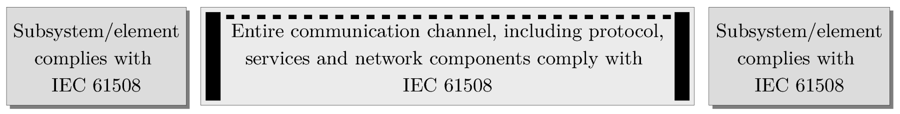
Table 2.1: Communication Profile Families (CFP) and Functional Safety Protocols defined by IEC 61784–3

2.1.1 Assumptions on the communication channel

The functional safety protocols defined by IEC 61784-3 have been designed assuming the underlying communication system behaves like a *black channel*. With such an approach, the safety nodes are not aware of the channel features, nor of the industrial protocol they rely on, except for the service primitives necessary to data transmission. Moreover, none of the error detection mechanisms of the underlying communication technology is employed to guarantee the integrity of the transferred process data. In this respect, with reference to Figure 2.3a, the communication (black) channel, as seen by the safety communication layer, begins at the boundary with the application layer. With this approach, as shown in Figure 2.4a, it is sufficient that only the interfaces and services towards the communication channel are designed according to IEC 61580 and IEC 61784-3. In this way, although the functional safety protocols of IEC 61784-3 have been designed for a specific network, as evidenced in Tab. 2.1, in principle, they could be implemented on whatever network.


Interfaces with communication channel, including services and protocols, complie with IEC 61784-3
Subsystem/element complies with IEC 61508
Part of the communication channel between the interfaces are not designed or validated to IEC 61508
Subsystem/element complies with IEC 61508

(a) Black channel


Subsystem/element complies with IEC 61508
Entire communication channel, including protocol, services and network components comply with IEC 61508
Subsystem/element complies with IEC 61508

(b) White channel

Figure 2.4: Representation of black and white channel taken from Ding et al. (2016)

Conversely, in Figure 2.3b and Figure 2.4b, the *white channel* approach assumes that the functional safety protocol is well aware of the underlying communication system, which must comply with IEC 61508, and uses its services and protocols to implement the safety functions. With such an approach, every component of the communication channel is considered safety–relevant, so that it has to be designed to fulfill the safety requirements of the specific application. Although this may seem like a complication, it simplifies the design of the safety communication layer. It is sufficient that this only integrates additional safety features to those already encompassed by the standard communication protocol to reach desired SIL. The major drawback of this design approach is that the components have an inflexible architecture, making the migration to newer technologies slow and costly.

Regardless of the channel approach, functional safety protocols have to be able to deal with several types of communication impairments. They are listed in Table 2.2. The functional safety protocols have to adopt adequate countermeasures against possible errors to achieve the residual error rates necessary to ensure the adequate Safety Integrity Level (SIL).

Error	Description
Corruption	The received message is corrupted
Repetition	The received message is the repetition of a formerly received one
Incorr. Seq.	The received message violates the sequencing rules
Loss	The message is not received by the recipient
Delay	The received message has an unacceptable delay
Insertion	The received message comes from an unexpected (invalid) source
Masquerade	The received message is not safety–relevant although it comes from a valid source
Addressing	The received message is safety–relevant, but it was intended for another destination

Table 2.2: Communication errors dealt with by functional safety protocols

In particular, IEC 61784-3 specifies eight countermeasures, as reported in Table 2.3.

The SIL is determined by a risk analysis that involves all the safety functions of a given plant, machinery, and other equipment. Notably, IEC 61508 recommends that the adopted communication facilities, e.g., functional safety fieldbuses, may influence with a maximum percentage of 1% the average frequency of dangerous failures per hour (PFH) of the safety functions (assuming that every single error could lead to a dangerous failure). This reflects on the performance figures that functional safety fieldbuses have to provide, in terms of Residual Error Probability (REP) per hour, as summarized in Table 2.4.

REP refers to the number of failed safety messages per hour and clearly depends on

- the bit error rate of the underlying channel;
- the number of safety messages transmitted per hour;
- the countermeasures against errors adopted by the functional safety protocols;

Countermeasure	Description
Sequence Number	Unique sequence number associated to each transmitted message
Time Stamp	Time stamp associated to each transmitted message
Time Expectation	Maximum tolerated delay between the arrival of two consecutive messages
Connection authentication	Unique identifier of the connection between sender and receiver of a message
Feedback message	Acknowledgment sent by the receiver of a message to the sender to confirm the correct reception
Data Integrity Assurance	Techniques adopted to detect possible corruption of safety data (typically redundancy checks)
Redundancy with Cross–Checking	Cross–checking of messages transmitted via redundant systems
Different Data Integrity Assurance Systems	Use of diverse data integrity techniques for safety–related data and non safety–related data
Addressing	The received message is safety–relevant, but it was intended for another destination

Table 2.3: Countermeasures adopted by functional safety protocols

SIL	REP [failed messages per hour]
SIL 1	$< 10^{-7}$
SIL 2	$< 10^{-8}$
SIL 3	$< 10^{-9}$
SIL 4	$< 10^{-10}$

Table 2.4: Safety Integrity Levels and Residual Error Probability

It may be observed that IEC 61784-3 makes assumptions on the underlying channel, as well as on the number of safety messages, that reveal severe for the performance required by the functional safety protocols (particularly for the countermeasures against errors). Indeed, it assumes that the functional safety fieldbuses are deployed in “high demand systems” so that the number of safety messages they have to deal with is high (although the term “high” is not clearly specified). In addition, IEC 61784-3 assumes that the bit error rate of the underlying (black) channel is equal to 10^{-2}, which definitely represents a high value, even for an error–prone environment like the industrial communication scenario (Willig and Wolisz (2001); Vitturi (2004); Sestito, Turcato, Dias, Rocha, da Silva, Ferrari, and Brandao (2018)).

2.2 Functional Safety Protocols based on the White Channel Approach

This section analyzes one of the protocols based on the white channel and, therefore, with a completely different design approach from the one encompassed by IEC 61784-3. We will illustrate the fundamental aspects in order to provide a helpful way of comparison.

2.2.1 CANopen Safety

CANopen Safety, standardized by EN 50325-5, is likely the most popular functional safety protocol that relies on the white channel approach. It is defined at the application layer of the well known Controller Area Network (CAN) (ISO 11898-1) and, as such, it can be used in conjunction with the CANopen application layer protocol (EN 50325-4).

The basic design idea of CANopen Safety is the transmission of the safety data in two subsequent legacy CAN frames, which constitute a CANopen Safety message called Safety Relevant Data Object (SRDO). As shown in Figure 2.5, the first CAN message contains the safety data; the second one conveys the same data but bitwise inverted. This concept, called serial redundancy, allows the use of the entire 8-byte data field of

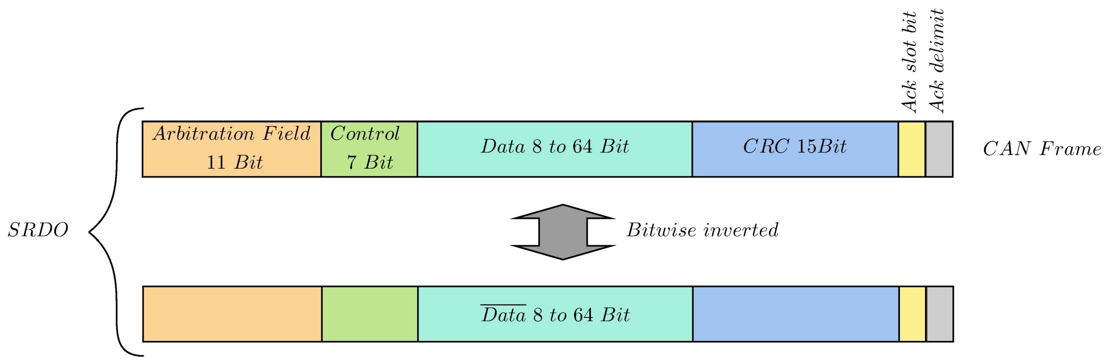


Figure 2.5: SRDO in CANopen safety.

the CAN frame. Moreover, its implementation appears relatively simple since, similarly to CANopen, it is based on the definition of Communication Objects (COBs).

From the above description, the white channel approach adopted by CANopen safety appears evident. It allows to exploit all the error detection techniques typical of CAN (e.g., CRC, bit stuffing, and ACK). Moreover, CANopen Safety uses additional countermeasures that derive from its own design features. Notably, upon receiving the two CAN messages that constitute an SRDO, each safety device crosschecks them. Also, a running number is used to detect possible safety message repetitions, loss, insertion, and wrong sequences.

The transmission of SRDOs occurs periodically by devices called "SRDO producers", whereas devices that receive SRDOs are called "SRDO consumers". This allows for a further error detection strategy based on the use of two timers. As shown in Figure 2.6, the

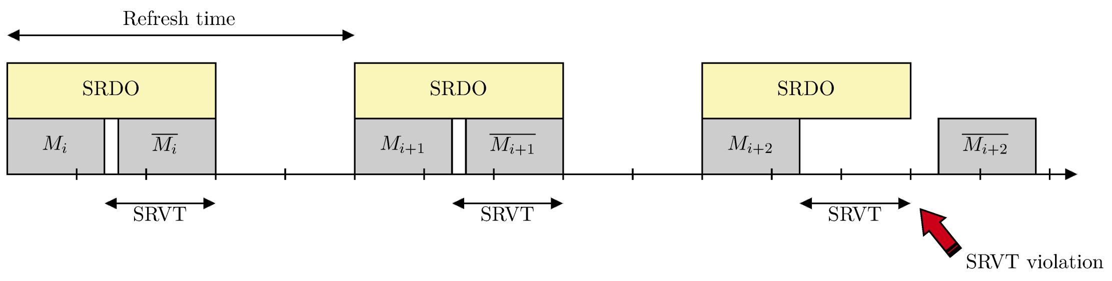


Figure 2.6: SRVT timing.

time interval between the transmission of the two CAN messages M_i and $\overline{M_i}$ composing an SRDO is called Safety Relevant Validation Time (SRVT). The SRDO consumer expects to receive $\overline{M_i}$ within SRVT. Otherwise, the SRDO is marked as invalid since it is not possible to perform the crosscheck. Similarly, another timer, called Safeguard Life Time (SCT), monitors the transmission period of an SRDO. As can be seen in Figure 2.7, if the complete SRDO does not arrive within the SCT, an error occurs. Both these times have nominal and maximum values. Generally, if a time–out occurs for any of the two timers, it leads to the activation of a safe state on the SRDO consumer.

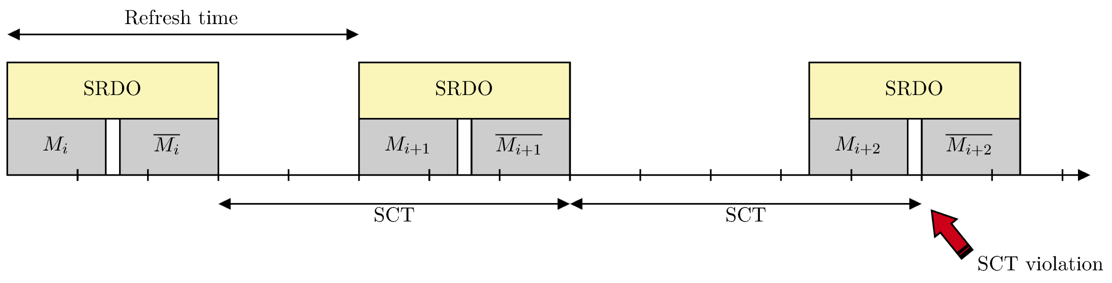


Figure 2.7: SCT timing.

The countermeasures adopted by CANopen Safety to detect communication errors can be summarized as follows.

- Running-number for all the SRDOs
- Timers for monitoring the correct SRDO scheduling.

- Acknowledgment of each CAN message which is part of a SRDO.
- Double identification of an SRDO via a dedicated identifier sequence (of bits) for the first CANOPEN Safety part containing the information and another one for the the second bit-wise inverted part.
- Crosscheck of the two CAN messages composing a SRDO bitwise inversion.
- CRC checksum to detect possible frame corruption, implemented at CAN level.

Thus, with refer to the communication errors identified by IEC 61784–3, Tab. 2.5 indicates the effectiveness of the aforementioned countermeasures.

Error	**Safety Countermeasures**					
	Runn. Nr	Timers	ACK	rx/tx SRDO	Crosschk.	CRC
Corruption			✓		✓	✓
Repetition	✓				✓	
Incorr. Seq.	✓				✓	
Loss	✓		✓		✓	
Delay		✓				
Insertion	✓		✓	✓	✓	
Masquerade				✓		
Addressing				✓		

Table 2.5: Safety measures adopted by CANopen Safety to detect communication errors

2.3 Functional Safety Protocols based on the Black Channel Approach

This section will discuss the fundamental characteristics of two FSF based on the black channel approach. The first one is OPC-UA Safety which is one of the protocols introduced by Industry 4.0. Although it is not yet regulated under IEC 61784-3, it is designed to meet its requirements. The second protocol we will analyze is Fail Safe over EtherCAT (FSoE). Thanks to its implementation based on the black channel, it will later be used as a case study for implementation on a wireless network.

2.3.1 OPC UA Safety

OPC-UA Safety is a protocol defined in agreement with the black channel approach, specifically conceived assuming OPC-UA as underlying communication protocol IEC 62541. The architecture of the OPC-UA Safety protocol is reported in Figure 2.8.

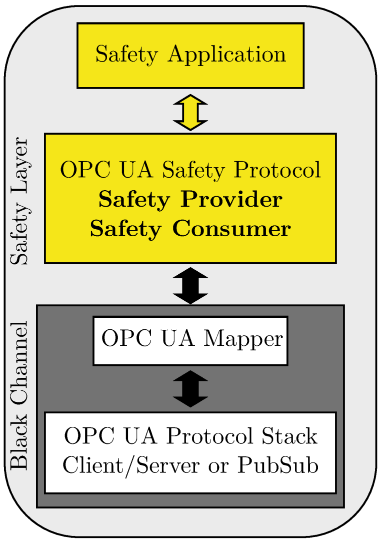


Figure 2.8: OPC-UA Safety Protocol Architecture

As can be seen, it is based on two services, namely Safety Provider and Safety Consumer, that represent the building blocks of the safety layer that are directly connected with the safety applications. The OPC-UA Mapper is a sub-layer responsible for the interface between the safety layer and OPC-UA necessary to transmit Safety Protocol Data Units (SPDUs). Indeed, both the Mapper and the OPC-UA protocol constitute the black channel. Notably, both the OPC-UA versions, referred to as Client/Server and Pub/Sub, can be used to transfer SPDUs. Thus, in particular, the availability of Pub/Sub allows adopting the real–time OPC-UA architecture described in Bruckner et al. (2019) that makes use of the Time-Sensitive Networking (TSN) family of standards (Lo Bello and Steiner (2019)). In this way, the performance figures of OPC-UA Safety will likely be comparable with those of the safety protocols designed to rely on Real-Time Ethernet (RTE) networks.

Safety data exchange takes place cyclically among Safety Producers and Safety Consumers, located on different network nodes (acyclic communication is also possible, but typically during the protocol initialization phase for parameter exchange). OPC-UA Safety has been designed to encompass several Producers and Consumers. Moreover, it is a "dynamic" protocol, in those safety nodes (and hence Safety Providers/Consumers) may be added/removed during operation. Safety Provider and Consumer have to establish a connection before starting data transmission. After that, the safety data are transmitted via the exchange of the request and response SPDUs described in Figure 2.9.

RequestSPDU

	SafetyConsumerID	MonitoringNumber	Flags
Nr. of Bytes	**4**	**4**	**1**

ResponseSPDU

SafetyData	Flags	SPDU_ID	SafetyConsumerID	MonitoringNumber	CRC	NonSafetyData
Up to 1500	**1**	**12**	**4**	**4**	**4**	**Up to 1500**

Figure 2.9: OPC-UA Safety Protocol Data Units

The Safety Consumer issues the request SPDU towards the Safety Producer. As can be seen, it comprises a field that identifies the requesting consumer (SafetyConsumerID) and another field (MonitoringNumber) which is used to detect possible SPDU duplications. The Flags field reports further status information. The response SPDU is issued by the Safety Provider as a response to the request. The first field contains the safety data encapsulated as OPC-UA variables. The Flags field reports status information, whereas the field *SPDU_ID* allows identifying the Safety Provider. The subsequent two fields are the copies of the correspondent ones transmitted with the request SPDU. The CRC field is used to check for possible data corruption. Finally, as can be seen, a response SPDU can also contain non–safety data.

The countermeasures against the communication errors adopted by OPC-UA Safety are listed below.

- Sequence Number. Both the Request and Response SPDUs exchanged by Safety Consumer and Provider contain the field MonitoringNumber. This value changes every time a new request SPDU is issued. A response SPDU is accepted only if the monitoring number of both request and response SPDUs has the same value.

- Time expectation. The Safety Consumer uses a timer to check whether a Response SPDU arrives on time. A timeout value is defined by a specific protocol parameter set during the initialization phase. The expiration of the timeout allows detecting a time expectation violation.

- Feedback message. This countermeasure is natively implemented by OPC-UA Safety thanks to the mechanism of SPDUs exchange. Indeed, each Request SPDU has to be acknowledged by the correct reception of the correspondent Response SPDU.

- Connection Authentication. The field *SPDU_ID* in the response SPDU is calculated by the Safety Provider each time a new response SPDU is transmitted. The Safety Consumer checks this value to detect whether or not the SPDU is arriving from the intended Safety Provider.
- Data integrity assurance. This measure is implemented employing the data integrity check achieved with the CRC, calculated by the Safety Provider, and checked by the Consumer upon reception of each Response SPDU.

Table 2.6 lists the countermeasures adopted by OPC-UA Safety, along with the communication errors they can detect.

Error	**Safety Countermeasures**			
	Mon. Nr.	Timeout	*SPDU_ID*	CRC
Corruption				✓
Repetition	✓	✓		
Incor.Seq.	✓			
Loss	✓	✓		
Delay		✓		
Insertion	✓			
Masquerade	✓		✓	✓
Addressing			✓	

Table 2.6: Safety countermeasures adopted by OPC-UA Safety to detect communication errors

2.3.2 Fail Safe over EtherCAT

Fail Safe over EtherCAT (FSoE) is referred to as Functional Safety Communication Profile 12–1 by IEC 61784-3-12. It has been conceived for deployment in conjunction with EtherCAT.

FSoE is a Master–Slave protocol, with a unique device referred to as FSoE master and several FSoE slaves. During normal operation, the FSoE master cyclically polls the FSoE slaves. The data exchange takes place over FSoE connections, which are virtual communication channels established between the FSoE master and each FSoE slave in the initialization phase.

The FSoE PDU has two different formats depending on the amount of safe data bytes that have to be exchanged. The simplest format is shown in Figure 2.10, and it is used to

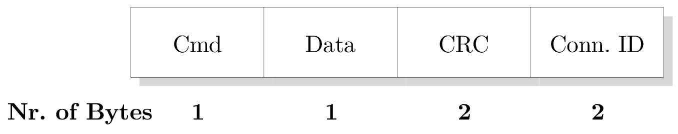


Figure 2.10: Basic FSoE frame

transfer a single byte of safety data from master to slave and vice versa. The first byte, referred to as *Command*, contains the specific state of the FSoE connection, allowing to determine the meaning of the safety data. This is followed by the *Data* field, which can be extended up to 2 bytes if the safety information cannot fit in the basic FSoE frame, followed by the 16-bit (2 bytes) *Cyclic Redundancy Check (CRC)*, and then by the *Connection Id* (2 bytes), which is a unique identifier of the current specific FSoE connection.

The CRC is calculated in a more elaborated way with respect to traditional protocols. Indeed, it is determined on a data structure, which includes more fields than those of the Safety PDU. They are reported in Tab. 2.7.

Field Nr.	**Field**
1	received CRC (bit 0-7)
2	received CRC (bit 8-15)
3	ConnID (bit 0-7)
4	ConnID (bit 8-15)
5	Sequence Number (bit 0-7)
6	Sequence Number (bit 8-15)
7	Command
8	SafeData[0]
9	0
10	0
11	0

Table 2.7: FSoE: Fields used for CRC Calculation

As can be seen, two fields refer to the sequence number. This is a 16–bit counter ranging from 1 to 65535, re–initialized to 1 when the maximum value is reached, representing the progressive number of the PDU transmitted by a safety device. Each safety device handles its own sequence number, representing the progressive number of the safety PDU it transmits. A device also maintains the sequence number of the Safety PDU it expects to receive from its partner. The comparison between these two values (expected to coincide) is achieved by inserting the sequence number in the structure used to calculate the CRC. Moreover, as described by IEC 61784-3-12, three additional null octets are included in the structure used for CRC calculation.

	Cmd	Data_0	CRC_0	...	Data_i	CRC_i	Conn. ID
Nr. of Bytes	**1**	**2**	**2**		**2**	**2**	**2**

Figure 2.11: Extended FSoE frame

The second PDU format, described in Figure 2.11, is used when more than a single byte of safety data has to be transferred.

As can be seen, it contains several blocks of two–byte data, each followed by the relevant CRC. The structure adopted for CRC calculation is modified accordingly.

During normal operation of the protocol, the FSoE master sends a Safety PDU to all the addressed slaves and waits for the answer. Notably, each slave may have a different elaboration time, so the response will take multiple EtherCAT cycles to be sent back to the Master. The state diagram of both FSoE master and slave is reported in Figure 2.12.

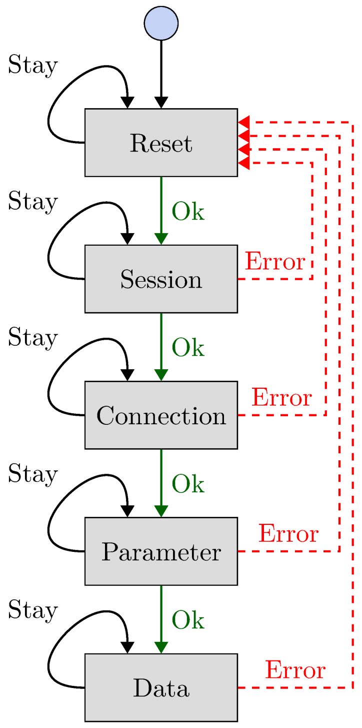


Figure 2.12: FSoE state machine of a slave device

After a power-on, both FSoE Master and Slave are in the “Reset” state, which is also considered the fail–safe state. The transition from one state to another is determined by the issue of a specific command, the successful exchange of safety relevant data, or by completing configuration procedures. The nodes move from the “Reset” state to “Session”, in which the devices mutually exchange unique random generated 16-bit Session ID, which allows differentiating multiple SPDUs sequences in the event of several restarts of the FSoE connection. In this way, SPDUs generated in a different session should be distinguishable. The nodes then move to “Connection” where the FSoE connection is established, upon suitable commands from the master. Similarly to the previous state, the devices mutually exchange the unique 16-bit Connection ID. This parameter establishes a one-to-one relationship between the FSoE master and the FSoE slave so that the SPDUs exchanged by the two are uniquely determined. Subsequently, in “Parameter”, the operational safety parameters are exchanged, and, finally, the master sends a command to enter the “Data” state in which the safety relevant process data is actually exchanged. In every state, an immediate transition to “Reset” may occur due to either a command received from the master or a problem detected by the slave.

In each device (master and slaves), a watchdog timer monitors the FSoE communication cycle to detect possible delays on the network. If the FSoE master does not receive the answer from a queried slave within a specified time–out, then the watchdog timer of the master triggers the re-initialization of the FSoE connection with that slave. Conversely, if an FSoE slave is not queried by the master within a time–out, then the watchdog timer of the slave forces such device to enter the reset state. In this state, the slave waits to be re-initialized by the master. The FSoE master can handle several slaves by establishing a unique FSoE connection with each of such devices.

The communication medium, from the safety point of view, is seen as a black channel. With such an approach, safety applications and standard applications can coexist, sharing the same communication system simultaneously (IEC 61784-3-12).

The safety protocol encompasses procedures to systematically detect faults or errors that could occur during operation. The types of faults considered in this work are classified in Tab. 2.8, which also lists the approach adopted by FSoE to detect their occurrences.

As an example, the corruption of a Safety PDU is detected by checking the CRC. Analogously, the loss is detected by a wrong sequence number as well as by the intervention of the watchdog timer.

	Safety Countermeasures			
Error	Con. Id	Seq. Nr	W.dog	CRC
Corruption				✓
Interchange	✓	✓		
Repetition		✓		
Insertion		✓		
Loss		✓	✓	
Delay			✓	
Misrouting	✓			

Table 2.8: Safety measures adopted by FSoE to detect errors and faults

3
The Fail Safe over EtherCAT (FSoE) protocol implemented on the IEEE 802.11 WLAN

In the era of Industry 4.0 and particularly the Industrial Internet of Things (IIoT), several manufacturing units, whose functions are dependent on one another, can be physically located in different sections within the production area or even in distinct places. This means that the cabling is increasingly complex and branched due to the ever-larger size of the plants and their distributed locations. Wires are difficult to install and maintain, and therefore the possibility of replacing them with wireless reconfigurable links seems a very attractive opportunity and a fundamental step to incorporate particular applications into the IIoT context (Frotzscher, Wetzker, Bauer, Rentschler, Beyer, Elspass, and Klessig (2014a); Refaat, Daoud, Amer, and Makled (2010a)). For example, as pointed out by Taylor, Akerberg, Ibrahim, and Gidlund (2012), traditional fieldbus communication fails to give a complete solution and efficient integration in contexts such as mobile autonomous collaborative robotics (Frotzscher, Wetzker, Bauer, Rentschler, Beyer, Elspass, and Klessig (2014b); Refaat, Daoud, Amer, and Makled (2010b)) or systems operating in adverse environmental conditions (Xu, Sun, Liu, Wang, Gu, and Liu (2020)). Driven by these reasons, wireless extensions of automation networks and fieldbuses have been researched in different forms (Vitturi, Carreras, Miorandi, Schenato, and Sona (2007); Åkerberg, Gidlund, Lennvall, Neander, and Björkman (2011)). Another challenge is using these systems in a safety critical context in which the malfunction of the communication system may lead to irreversible and catastrophic damages to people and the surrounding environment. Therefore in such applications, ensuring the correct transmission of information is a fundamental requirement. For example, Sha, Shi, and Watkins (2006) proposed the use of Wireless Sensor Networks (WSNs) for fire rescue applications. Ikram, Jansson, Harvei, Fismen, Svare, Aakvaag, Petersen, and Carlsen (2013) successfully exploited the black channel to implement a SIL compliant wireless hydrocarbon leak detection system based on ProfiSafe (IEC 61784-3-3) and ISA100.11a (IEC 62734). In Hashemian (2010) and Hashemian (2011), WSNs have been exploited to monitor safety critical devices in nuclear plants.

Moving to more general researches, Breiling, Dieber, and Schartner (2017) addressed the implementation of a safety and security layer on top of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) for the communication between nodes based on Robot Operating System (ROS). Both Åkerberg, Reichenbach, and Björkman (2010) and Åkerberg et al. (2011) and proposed a proof-of-concept for the use of WirelessHART (WirelessHART (2020)) in safety-critical communications based on ProfiSafe. They successfully exploited the black channel to integrate into the ProfiSafe network of wireless gateways to extend the wired network. Essentially, they only changed the transmission system without changing anything in the upper layers of the protocol stack. Yang, Ma, Xu, and Gidlund (2018) proposed a similar work focusing mainly on an enhanced WirelessHART scheduling to minimize the Safety Function Response Time (SFRT). Indeed, WirelessHART is based on Time Division Multiple Access (TDMA), and all devices are time–synchronized and communicate in prescheduled 10 ms time slots. The absence of asynchronous frames represents a bottleneck in cases where high priority frames, such as alarms, must be sent. This problem has been widely addressed with the proposal of several modified Medium Access Control (MAC) layers able to offer additional QoS features. PriorityMAC has been proposed by Shen, Zhang, Barac, and Gidlund (2014), which introduces a novel medium access method that enables higher priority traffic to hijack the dedicated transmission bandwidth of the low priority traffic. Liang, Chen, Priyantha, Liu, and Zhao (2014) proposed RushNet, a framework able to prioritize traffic in multi-hop sensor networks. Zheng, Gidlund, and Akerberg (2016) proposed WirArb. This MAC, specifically conceived for industrial WSNs, supports multiple users by pre-assigning a specific arbitration frequency to each node, allowing to decide the order of channel access. With this mechanism, WirArb ensures that the node with the highest priority will immediately gain channel access, guaranteeing a deterministic behavior.

These solutions certainly improve the networks' responsiveness and reliability but they rely on ad-hoc non standardized solutions which are deeply in contrast with the IIoT paradigm (Li, Tang, Chan, Wei, Pu, Jiang, Li, and Zhou (2018a)). To successfully embody Industry 4.0, the proprietary solution must be replaced by open and standardized solutions (Weyer, Schmitt, Ohmer, and Gorecky (2015)). For this reason, in our work, we decided to follow a more conservative approach. Indeed, in our contribution, we propose an implementation of the Fail Safe over EtherCAT (FSoE) protocol on top of IEEE 802.11 WLAN. In this way, this implementation can be adopted in a large variety of devices equipped with a WiFi interface. Moreover, the future introduction of TSN on WiFi (Adame, Carrascosa, and Bellalta (2020b); Cavalcanti, Bush, Illouz, Kronauer, Regev, and Venkatesan (2020); Lee, Bernhard, Cavalcanti, Pang, and Val (2020); Sudhakaran, Mageshkumar, Baxi, and Cavalcanti (2021)), will bring seamlessly all the advantages introduced by the custom MAC layers but with the advantage of high interoperability.

3.1 Introduction

Although wireless networks have greatly improved, their performance figures are not yet comparable to those of wired industrial counterparts such as Profinet or Ethercat. Indeed, such networks are able to ensure cycle times below 100 μs (Orfanus, Indergaard, Prytz, and Wien (2013); Robert, Georges, Rondeau, and Divoux (2012); Prytz (2008)), a value that (for the time being) can not be achieved by any industrial wireless system. However, industrial networks used for the transmission of safety data have, to a certain extent, distinct requirements from those of typical factory automation applications. Indeed, in a master-slave communication implemented by a safety protocol, it has to be ensured that the master transfers safety data without corruption and that the response data from the slave are returned within a specified timeout, typically in the order of hundreds of milliseconds. Such performance figures can be provided by industrial wireless networks, particularly by the IEEE802.11 Wireless LAN (IEEE 802.11). This suggests that safety protocols can be, in principle, implemented over wireless networks, thus opening the field towards further employment of wireless networks in the context of factory automation Vitturi et al. (2018).

This section considers such a topic and describes an example of practical implementation of the popular FSoE safety protocol over the IEEE802.11 WLAN. Specifically, we first provide some implementation details and then describe the results of some practical measurements carried out on some different experimental setups.

3.2 FSoE Implementation

The FSoE stack has been developed and implemented on three different devices: a Personal Computer running the Linux operating system, a Raspberry Pi board, and an Espressif ESP8266 ultra low cost System on Chip (SoC) module. All such devices have Wi–Fi interfaces and are equipped with the full TCP/IP protocol stack. The implementation of FSoE has been made on top of the User Datagram Protocol (UDP). In practice, FSoE frames are encapsulated within UDP frames as described in Figure 3.1.

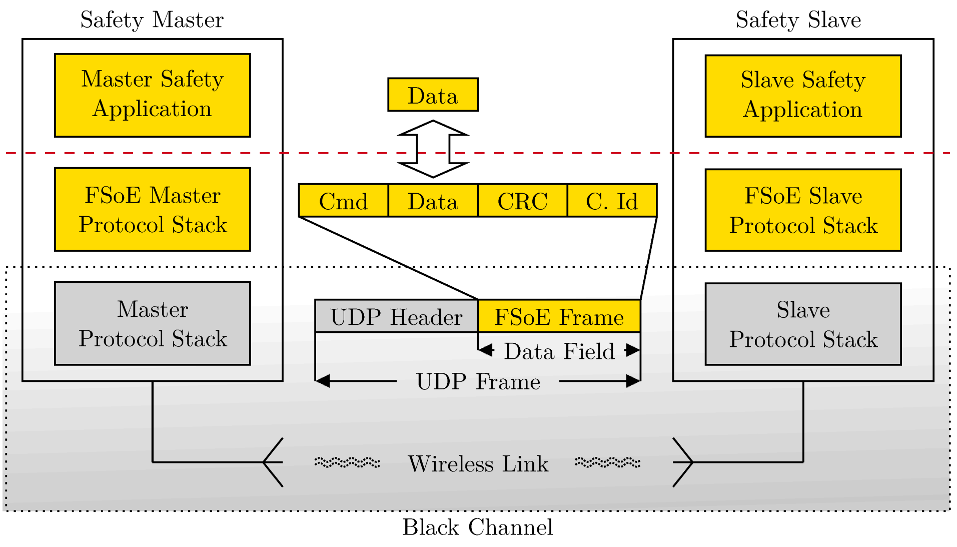


Figure 3.1: Implementation of the FSoE Protocol Stack

With this type of implementation, the black channel comprises not only the communication medium but also the Wi–Fi modules and the protocol stacks of both master and slave, up to the interface with the FSoE stack. As shown in Figure 3.1, the transmission of safety data is started by a safety application running on the master that issues a request to the FSoE protocol stack. The arrival of the safety data is notified via an indication primitive to the safety application of the slave, which issues the response primitive carrying the response data. The service is then concluded with the arrival of the confirm primitive to the master. Clearly, if during the elaboration within a node (either master or slave) a packet corruption occurs, data are protected by the CRC of the FSoE protocol, which detects the problem.

The FSoE master has a protocol instance for each slave. The master assigns a Connection ID to this protocol instance and associates it with the IP address of the slave. In this way, the master establishes a one-to-one relationship with the slaves so that a single UDP frame contains a single FSoE frame uniquely associated with a specific slave.

This type of addressing allows obtaining a double check of the correct addressing of the slave thanks to the correspondence between the IP address and connection ID. Indeed, the slave knows both its connection ID and IP address, and, therefore, upon receiving a frame, it can check whether the addressing is correct. Similarly, upon receiving a frame, the master can immediately check the matching between IP address of the slave and connection ID. Also, the master can check the possible duplication of Connection IDs. If any mismatch is detected, the FSoE Master resets the FSoE Connection and forces a re-initialization.

As an alternative, a different way of exchanging safety data could be devised. With this solution, the safety data addressed to all slaves, prepared by the FSoE protocol, are delivered via a single broadcast UDP frame. Such a frame is composed of several fields, each of them associated with a specific slave. Then each slave answers with a unicast communication frame to the master. The access to the channel by the slaves may be regulated by a sequence specified in the frame sent by the master. In this way, each slave knows when it can transmit the response data. However, also leaving the slaves to access the channel randomly might be a viable strategy. Indeed, the FSoE protocol does not impose a strict deterministic procedure on data exchange but, rather, it requires that safety data arrives within a specified time–out. Although this alternative way has not been implemented, it looks appealing from both a communication and a computational delay aspect, which are particularly important for resource constrained setups. Indeed, it dramatically reduces the number of exchanged UDP frames (for example, with N slaves, in the former case, $2N$ UDP frames are required, whereas, with the alternative strategy, they could be reduced to $N + 1$). Moreover, the overall elaboration time might be reduced as well, since all slaves receive data from the master simultaneously, and then they can prepare the response data in parallel.

It has to be noted that the safety protocol does not encompass any security mechanisms, possibly exposing the transmissions to common threats as for example spoofing or reply attacks. Nevertheless, reply attacks are rather difficult to implement since FSoE is designed in such a way that two successive frames should differ by at least one bit. As seen in Table 2.7, this is ensured by the use of successive sequence numbers as well as CRC. Because of this property, if two identical frames are received, they are marked as duplicates leading to the reset of the FSoE connection forcing the system to a safe been. Regarding spoofing, FSoE is implemented to accept or reject the frames based on the CRC. To successfully implement a spoofing attack, the attacker has to be able to change data and recalculate the CRC on the fly. This is possible only knowing exactly the current sequence number, which is not transmitted within the frame so the attacker must have monitored all transmissions since the initialization of the connection, and the previously received CRC. Another way would be by crafting a malicious data field in such a way that the CRC still remains valid but, due to the rather long polynomial used in the calculations, the probability of being able to exploit possible CRC collisions is practically very low. Although FSoE is potentially vulnerable to these types of attacks, the structure with which the protocol is implemented makes them computationally heavy and therefore difficult to implement.

3.3 Timing Analysis

In practical applications, the delivery of Safety PDUs from the master occurs cyclically with a period, T_{cycle}, determined by the application's requirements. Since all the slaves must be queried within a period, the polling time of a single slave represents a meaningful index of the protocol performance.

3.3.1 Polling Time of a FSoE Slave

The polling time of a slave has been calculated as the time that elapses between the generation of a FSoE frame transmission request by the master and the reception of the confirm primitive from the slave. As can be seen in Figure 3.1, the transmission sequence includes the times necessary to

- execute the FSoE protocol stacks in both master and slave;
- execute both master and slave standard protocol stacks;
- transmit the frame from master to slave and vice–versa.

Under the assumption that the stack execution times are the same for both master and slave, the polling time T_p can be expressed as

$$T_p = 2T_{FSoE} + 4(T_{stack} + T_{MAC}) + 2T_{tx} \tag{3.1}$$

where T_{FSoE} is the time to execute the FSoE protocol stack, T_{stack} is the time to execute the protocol stack, T_{MAC} is the execution time the last two bottom layers of the protocol stack, and T_{tx} is the time to transmit a frame. It is worth remarking that the term relevant to the execution time of the FSoE stack in Eq. 3.1 is $2T_{FSoE}$ since the generation of both the response and confirm primitives is made automatically by the stack, i.e., the FSoE protocol stack is executed only once at both the master and the slave units.

3.3.2 Time Slot Allocation to FSoE Slaves

The FSoE protocol has been implemented assuming that each slave may enter/exit the network dynamically. This implies that, periodically, the master starts a discovery procedure to identify the N connected slaves. Then, the master assigns a time–slot to each slave determined as

$$t_{slot} = \frac{T_{cycle}}{N} \tag{3.2}$$

In case the polling of a slave is carried out within t_{slot}, the master waits until the expiration of the whole t_{slot} and then moves to poll the next slave. Conversely, if there is a time–out (i.e. the polling has not been concluded within t_{slot}), the master forces the slave into the "Reset" state.

3.4 Experimental Results

Two different configurations have been adopted in the experimental sessions we carried out. The first one, referred to as "Basic Configuration", comprised one FSoE master and one FSoE slave, whereas the second one, referred to as "Multiple Configuration", comprised three FSoE slaves. In both configurations, we measured the polling time of the slaves and the percentages of lost packets, reflecting the number of FSoE connection re–initializations. The node acting as a master is configured as an IEEE802.11 Access Point (AP), whereas the slaves are configured as IEEE 802.11 Stations (STAs). In this way, they can connect directly to the FSoE master.

3.4.1 Basic Configuration Results

In the Basic Configuration, two different setups were deployed operating in the 2.4 GHz band. We used only ESP8266 modules in the first one, and the wireless connection was implemented by IEEE 802.11g at 54 Mbits/s. In the second setup, the FSoE master was a PC running the Linux operating system, and the FSoE slave was a Raspberry Pi board. In this second case, IEEE 802.11n was used at a rate of 72 Mbits/s. The experimental sessions, in both configurations, lasted 5 hours and implied the transmission of more than 9,000,000 packets in total. The experiments were executed in an industrial environment (the laboratory of a factory), where interference from other networks was monitored, and the IEEE 802.11 channel was selected to limit such a phenomenon. In both the experimental setups, nodes were located at a distance of 1 m. Figure 3.2(a) reports the

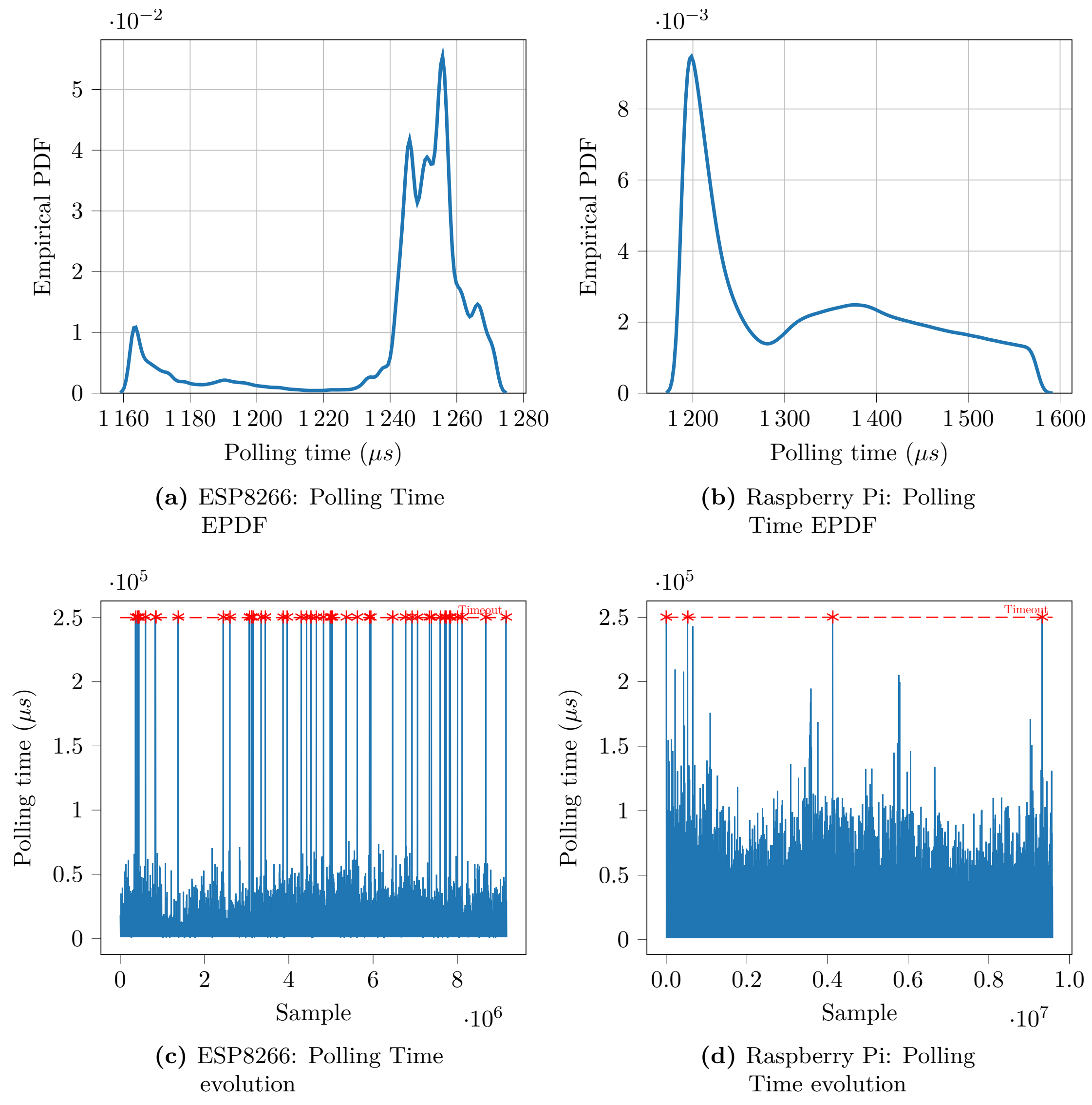


Figure 3.2: Polling time EPDF for the “Basic Configuration” scenario: (a) ESP8266 modules and (b) PC–Raspberry Pi setup. Comparison of polling time per sample, where the dashed red line represents the time-out value and star markers highlight lost packets: (c) ESP8266 modules and (d) PC–Raspberry Pi setup.

EPDF of the polling time for the ESP8266 setup[1], whereas Table 3.1 (top) shows its statistics.

Moreover, Table 3.1 (bottom) reports the performance in terms of lost FSoE packets.

[1] To ease the readability of the plots and the comparison among different configurations, Figure 3.2(a)-(b) report data comprised in the interval median ± one standard deviation. Also note that a different y-axis scale is used between (a) and (b), for the same reason.

Polling time [μs]				
mean	std	min	median	max
1484.67	1378.95	1051.00	1246.00	76004.00

Packet loss			
Total pkt	pkt lost	% pkt lost	test duration [h]
9169850	45	0.000491	5

Table 3.1: Statistics for ESP8266 modules: Polling Time (top) and Lost FSoE Packet (bottom)

Although there are no specific requirements for the polling time in the FSoE standard, the measured values can be considered suitable for a wide number of applications. Conversely, the percentage of lost FSoE packets is definitely not tolerable since the SIL3 requirement specifies as a necessary condition that at most one FSoE connection can be re–initialized once every 5 hours. In our experiments, 45 re–initializations occurred since each lost packet corresponds to one re–initialization. It should be noted that increasing the timeout duration would not have resulted in a reduction in the number of lost packets because these are packets that were never received and not packets received after the timeout deadline.

In order to investigate such behavior, we carried out additional tests, using the second setup in which the two ESP8266 modules were replaced by, respectively, a PC and a Raspberry Pi board. The results of the experiments are listed in Figure 3.2(b), which report the EPDF of the polling time, whereas Table 3.2 reports the polling time statistics and the percentage of lost FSoE packets.

Polling time [μs]				
mean	std	min	median	max
1592.07	1308.08	1096.52	1301.14	17942.40

Packet loss			
Total pkt	pkt lost	% pkt lost	test duration [h]
9584522	10	0.000104	5

Table 3.2: Statistics for PC–Raspberry Pi setup: Polling Time (top) and Lost FSoE Packets (bottom)

As can be seen, the values of the polling time have slightly incremented with respect to the ESP8266 setup and the EPDF more spread out. This is likely because, differently from the ESP8266, both the PC and the Raspberry Pi board use a general purpose operating system which may impact the whole execution times of the implemented protocol stacks. Conversely, the percentage of lost packets is dramatically reduced. Such results allow arguing that the number of lost packets is mainly determined by the performance of the Wi–Fi modules, which are of lower quality in the (cheap) ESP8266 modules.

These considerations are also confirmed by the lower panels of Figure 3.2 (c)-(d) that report the polling time behavior over the experiment for, respectively, the ESP8266 and the Raspberry Pi setups. Here the timeout values that determine the lost packet fraction are also reported, thus proving the different behavior between the two configurations. In addition, the IEEE 802.11 protocol stack in the PC–Raspberry Pi setup is configured to make use of the optional Request To Send (RTS)/Clear To Send (CTS) mechanism: clearly this allows to reduce the frame collision during the transmission and thus to obtain a lower packet loss rate compared to the ESP8266 setup, in which this feature could not be enabled.

3.4.2 Multiple Configuration Results

To better assess the (relatively) poor performances of the ESP8266, a test with multiple slaves has been carried out. Only ESP8266 modules were used in this configuration, and the network comprised one FSoE master and three slaves.

The results of the experimental measurements are listed in Table 3.3, which reports the statistic of the globally lost packets as well as the details for each FSoE slave.

	Packet loss		
	Total pkt	pkt lost	% pkt lost
Global	9713527	2206	0.022710
Slave 0	9713527	567	0.005837
Slave 1	9713527	894	0.009203
Slave 2	9713527	745	0.007669

Table 3.3: Lost FSoE Packet Statistics for multiple ESP8266 modules

It is evident that with multiple slaves, the percentage of lost packets is even more significant than that of the single slave setup, which may be due to the higher demand, in terms of system resources, to handle multiple slaves, confirming the inadequacy of such modules for a safety application.

3.4.3 Discussion

In all the analyzed setup, the percentage of lost packets is not acceptable by FSoE. However, some techniques to mitigate such a problem are available. For example, the transmission speed could be reduced, since lower rates adopt more robust modulations. Another possibility relies on the use of rate adaptation techniques. In this case, the transmission speed is selected according to the quality of the wireless link. Since the purpose of this work is to experience the wireless capability of the FSoE and its limitations, the implementation of such features is left to future work. Furthermore, during the experimental sessions, the FSoE nodes only exchanged SPDUs. In practice, however, the devices may need to transmit data in addition to the failsafe protocol, which will result in an even higher percentage of lost packets due to increased generated traffic. Also the analysis of such more complex scenarios is reserved for future work.

On the other hand, it is worthwhile to note that, although the tested configurations reveal not strictly compliant with the regulations, nonetheless it has been proved that it is possible to transfer safety PDUs successfully based on FSoE using IEEE 802.11 WLAN. This actually suggests that these implementations can be used in those less demanding applications where a connection re–initialization that occurs more frequently than once every 5 hours is acceptable. Nevertheless, the applicability of the proposed architecture must be evaluated and validated through safety assessments case by case.

As a further observation, Table 3.4 reports the execution time of the FSoE stack protocol for the ESP8266 module and the Raspberry Pi board, respectively. As can be seen, thanks to the larger computational capability, the Raspberry Pi board can elaborate an FSoE frame more quickly with respect to the ESP8266 module. However, for both setups the FSoE stack protocol execution time gives only a minor contribution to the polling time that is primarily due to the execution of the protocol stack.

	FSoE execution time [μs]				
	mean	std	min	median	max
ESP8266	50.63	3.06	47.00	50.00	111.00
Rasp.Pi	1.28	1.90	0.34	0.94	140.89

Table 3.4: FSoE protocol stack execution time for the ESP8266 and the Raspberry Pi boards

Finally, considering the statistics of the polling time EPDF for both the ESP8266 and Raspberry Pi setups, we aim to find the minimum time slot in order to allow at most one lost packet every 5 hours. Let be N_{frame} the number of frames exchanged in 5 hours. We want to calculate t_{slot} such that

$$P[T_p > t_{slot}] = 1 - P[T_p \leq t_{slot}] \leq \frac{1}{N_{frame}} \tag{3.3}$$

where T_p is the polling time and t_{slot} is the time slot. It results that the minimum time slot t_{slot} has to be 8621 µs and 16 084 µs, respectively, for the ESP8266 and Raspberry Pi setups. Hence, with a cycle time of 250 000 µs it is possible to connect at most 28 slave devices to the ESP8266 master device and 15 devices to the Raspberry Pi setup.

If we consider Eq. 3.1, a further evaluation of the experimental timings can be drawn. The transmission time T_x (in µs) can be obtained from Tramarin, Vitturi, Luvisotto, and Zanella (2016) as

$$T_x = 20 + 4\left\lceil 2\frac{36 + 2.75 + l}{r} \right\rceil + 6 \tag{3.4}$$

where $l = 43$ byte is the data length presented to the 802.11 layer, and r is the data rate (in $[Mbit/s]$), which is $54 Mbit/s$ for the ESP8266 setup and $72 Mbit/s$ for the Raspberry Pi setup: it results in $T_x = 38.11$ μs and $T_x = 30.54$ μs for the two setups, respectively. Giving that for the ESP8266 t_{stack} has been calculated experimentally in Bertolotti and Hu (2011) as equal to 48 µs, and for the Raspberry Pi it results in 35 µs according to Chuanxiong and Shaoren (2000), it follows that the T_{MAC} value is around 220 µs and 270 µs for the two considered configurations.

3.4.4 Conclusions

The experimental campaign on three different setups (single safety slave ESP8266 module, PC to Raspberry Pi board, and multiple safety slave ESP8266 modules) has evidenced similar results for the polling time, which results in about 1.5 ms. On the other hand, packet loss results are rather different for the three setups and, more importantly, do not comply with the limit imposed by the regulation in none of the experiments. In particular, at the best, a ratio of 10 lost packets every 5 hours has been achieved, while the regulations impose a maximum rate of 1 lost packet every 5 hours.

In this context, it can be pointed out that the proposed setups of wireless-FSoE can be used in less demanding application scenarios. However, current research activities focus on introducing suitable techniques to improve the quality of the wireless links and minimize the packet loss rate so that the full specifications for safety applications can be met.

4 Assessment of different Transport Layer protocols for Functional safety over Wireless

In a first analysis, the implementation we discussed so far of FSoE over IEEE802.11, revealed the significance and effectiveness of the proposed approach, particularly concerning the applicability of that implementation for some specific application areas. Besides, the performance achieved in terms of packet loss was still not sufficient to meet the strict FSoE requirements.

As knows, UDP is a connection–less protocol that does not encompass any QoS mechanism in which the successful delivery of a PDU entirely relies on the robustness of the transmission medium. Indeed, due to the lack of flow control and retransmission mechanisms in UDP, when a frame is lost or corrupted, there is no possibility of requesting its retransmission. In this session, we discuss the usability of different transport layer protocols to carry SPDU and their impact on polling time and number of lost packets.

4.1 TCP

The most straightforward approach to improving packet loss is to replace UDP with TCP. Indeed, unlike UDP, TCP is a connection-oriented protocol that intrinsically provides Quality of Service (QoS) features, such as error detection, flow control and re-transmission of undelivered packets. Exploiting the same approach adopted for the UDP implementation, as shown in Figure 3.1, FSoE frames are encapsulated into the TCP frames. This represents a seamless solution since the interfacing primitives between FSoE and TCP are practically identical to the ones used for UDP.

4.2 UDP with caching layer

Another approach is the use of UDP with an additional caching layer. As shown in Figure 4.1, the additional intermediate layer acts as an interface between UDP and the FSoE stack. The primary purpose is to enhance the protocol stack by introducing the handle of both duplicated and retransmissions of undelivered frames, which UDP does not conceive.

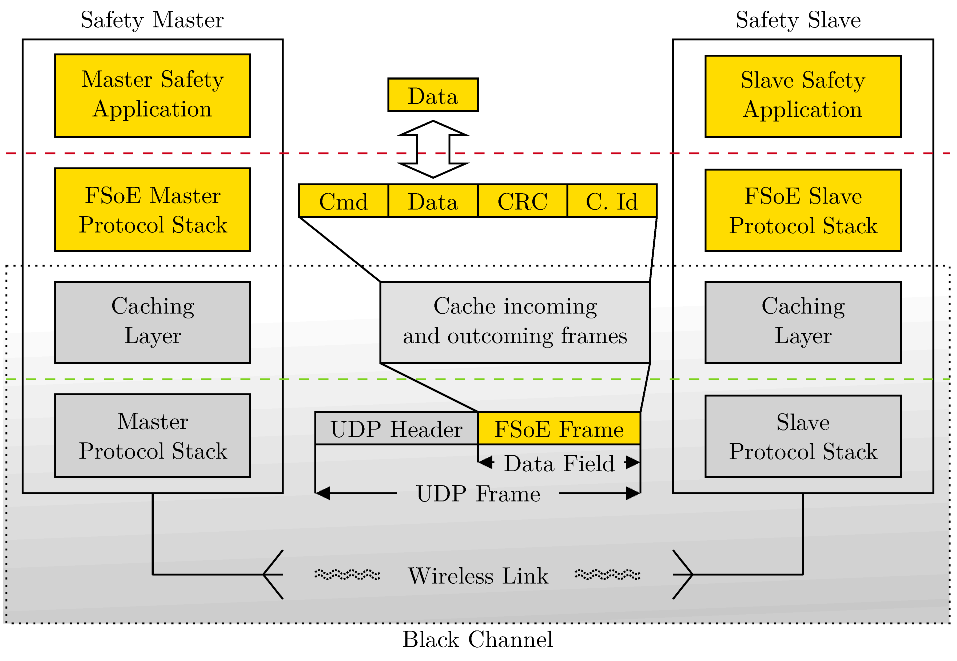


Figure 4.1: Implementation of the FSoE Protocol Stack on top UDP with addition caching layer

As can be seen in Figure 4.1, it is implemented on both the master and the slave. Each caching layer stores both the incoming from the protocol stack and outcoming SPDU from the FSoE stack. The caching layer expects to receive an incoming frame within a predefined deadline T_D for every outcoming frame. As shown in Figure 4.2, in case of packet loss, when the deadline T_D expires, the caching layer performs a retransmission of the stored outcoming frame. Notably, the deadline is completely independent of the FSoE watchdog that will continue to monitor autonomously for communication delays. Indeed, it will trigger the transition to the fail-safe state, i.e., the reset of the FSoE connection, if a valid SPDU is not passed to the FSoE stack within the preset watchdog time. Let be T_W the FSoE watchdog period and T_p the polling time of a device. To effectively exploit the retransmission mechanism provided by the caching layer, it must be

$$T_p \leq T_D < T_W \tag{4.1}$$

By selecting $T_D \geq T_W$, the caching layer would have no effect because the FSoE watchdog would expire before any retransmission could occur, thus obtaining the same behavior as a standard UDP connection.

A possible approach to select the retransmission deadline would be by imposing

$$T_D = \frac{T_W}{M}, \quad M > 1 \tag{4.2}$$

where M represents the maximum number of retransmissions that the caching layer will perform within an FSoE watchdog period. Moreover, the optimal value of M could be statistically determined (Khan, Qiu, Bhartia, and Lin (2015)) to fulfill the requirements on the maximum number of resets of the FSoE connection.

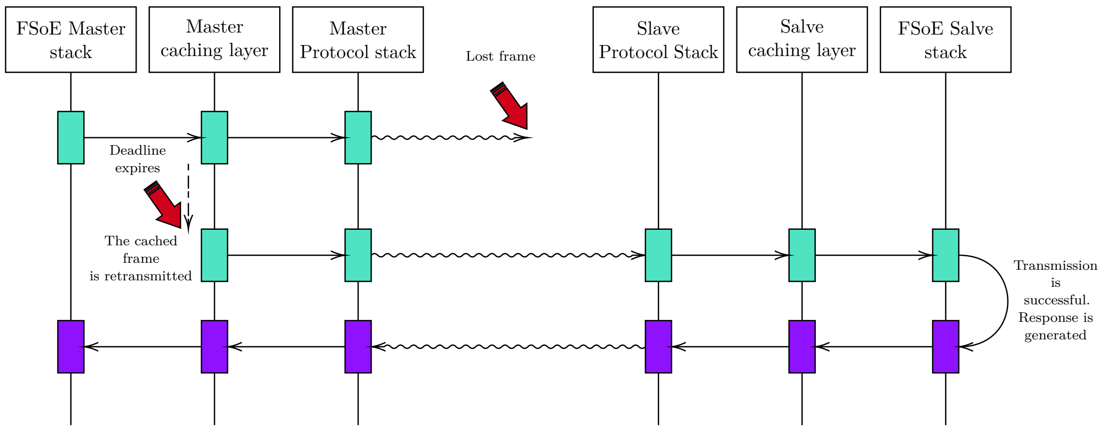


Figure 4.2: Handling of retransmissions in the caching layer

On the other hand, condition $T_p \leq T_D$ is not binding and has been imposed to minimize possible collisions between subsequent packets due to the saturation on the communication channel. In fact, the non respect of such constraint does not compromise the operation of the caching layer but only may lead to the generation of duplicated frames that, however, is a situation managed by the caching layer as well. Notably, generating duplicate packets is a condition to be avoided since, as specified in Section 2.3.2, it would result in the FSoE connection being reset.

The handling of duplicate frames is done by exploiting one of the fundamental features of SPDUs. FSoE is designed to ensure that two consecutive frames should differ at least one bit. This is ensured by the use of consecutive sequence numbers as well as CRC, as explained in Table. 2.7. Due to this characteristic, if two identical frames are received, those are certainly duplicated frames. In this situation, as can be seen in Figure 4.3, the duplicated frame is detected by caching layer by comparing the received frame with the previously stored incoming frame. Consequently, the previously stored outcoming frame is sent without executing the FSoE stack.

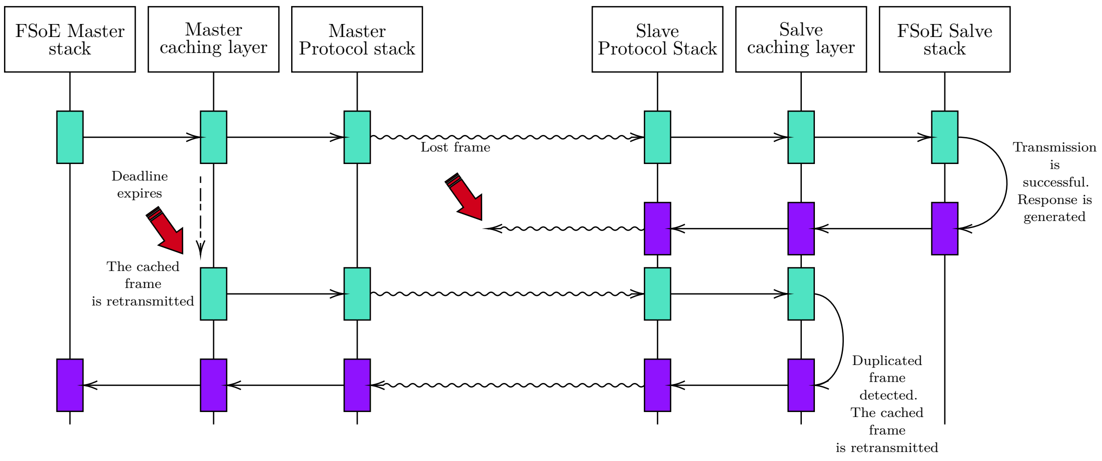

FSoE Master stack
Master caching layer
Master Protocol stack
Slave Protocol Stack
Salve caching layer
FSoE Salve stack
Lost frame
Transmission is successful. Response is generated
Deadline expires
The cached frame is retransmitted
Duplicated frame detected. The cached frame is retransmitted

Figure 4.3: Handling of duplicated frame in the caching layer

The working principle of the caching layer is summarized in Algorithm 1. The layer continuously polls the transport layer for an incoming frame (line 2), which is eventually acquired (line 3). The incoming frame is compared against the previously stored incoming frame (line 4), and in case they differ, the FSoE stack is executed, and both the generated response and incoming frame are locally stored for future comparisons (lines 6-8). The handle of duplicated frames is implemented by lines 11-12. Indeed, if the incoming frame matches the one received on the previous transmission, the caching layer responds with the stored outcoming frame without triggering the execution of the FSoE stack, thus avoiding the reset of the FSoE connection. When there are no incoming frames, the algorithm monitors whether the response's deadline is exceeded (line 14). If so, then the previous outcoming frame is retransmitted (line 15). Moreover, lines 19-21 handle the callbacks from the FSoE stack in case a watchdog violation occurs and allow the device to be forced into a safe state.

Algorithm 1: Caching layer

```
while true do
    if has imcoming frame from UDP then
        incomingFrame = ReceiveFsoeFrameFromUdp();
        if incomigFrame != storedIncomingBuffer then
            reset deadline $T_D$;
            storedIncomingBuffer = incomingFrame;
            outcomingFrame = executeFsoeStack(incomingFrame);
            storedOutcomingBuffer = outcomingFrame;
        else
            outcomingFrame = storedOutcomingBuffer;
        end
        TrasmittFsoeFrameToUdp(outcomingFrame);
    else
        if deadline $T_D$ is expired then
            TrasmittFsoeFrameToUdp(storedOutcomingBuffer);
        end
        update deadline;
    end
    update watchdog;
    if watchdog is expired then
        trigger fail safe condition;
        reset FSoE stack;
    end
end
```

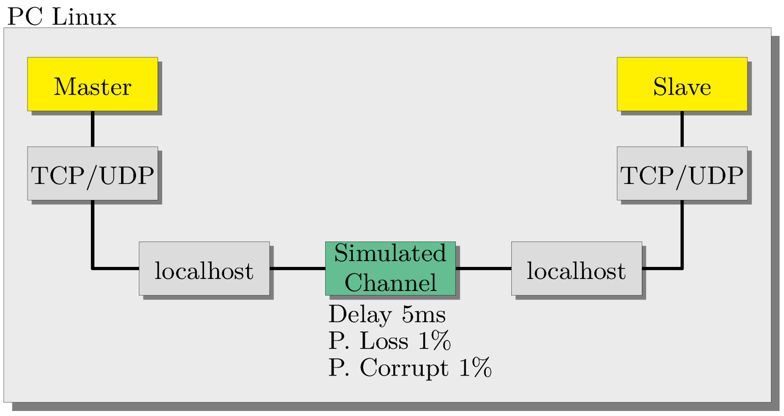


Figure 4.4: Experimatl setup to evaluate the trasport layer on localhost

4.3 Experimetal Setup

In order to evaluate the performance in terms of polling time T_p and packet loss, all the three variants of the transport layer, namely UDP, TCP, and UDP with caching layer (UDPc), were implemented and test on two different experimental setups. The first one, named Simulated Channel configuration, is represented in Figure 4.4. The three implementations have been tested on a PC running Linux, which executes both the Master and Slave stack instances and exchanges packets through a localhost link, i.e., without a physical link layer. This experimental session has been performed to provide a meaningful and fair assessment without the influence of external factors. The simulation of channel impairments such as delay and packet loss has been achieved using the kernel tool `TC (Traffic Control)`, which allows shaping, scheduling, policing, and dropping network traffic in any network interface. Exploiting this tool, we set up a rule[1] to simulate a network that introduces a constant delay of 5 ms and present a Packet Loss Rate (PLR) and a Packet Error Rate (PER) of 10^{-2}. Notably, the selected PLR and PER are the maximum conceived by IEC 61784-3-12.

The second configuration, named IEEE802.11 Link, reproduces the one presented in section 3.4.1 but in a noisier environment, where a Raspberry Pi and a PC exchange SPDUs via a wireless link.

In both configurations, a watchdog timeout T_W of 250 ms has been set as the time limit between the reception of two consecutive SPDUs, while the reception deadline T_D in the caching layer has been set to 5 ms. With the selected values of T_W and T_D, in case of packet loss, the number of retransmissions that the caching layer will perform before a watchdog violation is 50. Moreover, since the simulated transmission channel introduces

[1]`tc qdisc add dev lo root netem delay 5ms loss 1% corrupt 1%`

a constant delay equal to 5 ms, the condition $T_p \leq T_D$ will undoubtedly be violated, thus introducing a considerable number of duplicate packets.

4.4 Results

4.4.1 Simulated Channel configuration

The first set of outcomes is shown in Table 4.1, which reports the total number of exchanged packets and the corresponding number of lost frames. As can be seen, the results obtained for UDP confirm those obtained in Section 3.4. UDP, not conceiving any flow control mechanism, in the presence of channel impairments, neither the retransmission of lost frames nor the handling of any duplicated ones is guaranteed. Conversely, both TCP and UDPc, guarantee the successful delivery of all transmitted SPDUs, thus ensuring that the FSoE connection does not reset.

	Packet loss		
	Total pkt	pkt lost	% pkt lost
UDP	969654	31804	3.28
TCP	942469	0	0
UDPc	993966	0	0

Table 4.1: Lost FSoE Packet Statistics for different transport layer on simulated channel with both PLR and PER 10^{-2} and constant delay of 5 ms

The EPDF of the polling time for the tests carried out on the three versions of the transport layer is shown in Figure 4.5, whereas the statistics are reported in Table 4.1.

	Polling time (µs)			
	mean	std	min	max
UDP	10375.99	101.43	10148.09	10638.00
TCP	10428.83	102.56	10226.70	10729.79
UDPc	10376.49	106.49	10180.79	10686.70

Table 4.2: Polling time statistics for different transport layer on simulated channel with both PLR and PER 10^{-2} and constant delay of 5 ms

It should be noted that, since the simulated channel introduces a constant delay, all the randomness present in the measures are due mostly to the different software components used in the three implementations. Therefore, the polling time measure directly represents the performance of the transport protocol used, without the influence of external factors. Moreover, statistics and EPDF only take into account individual packets, i.e., lost or duplicate packets are not considered.

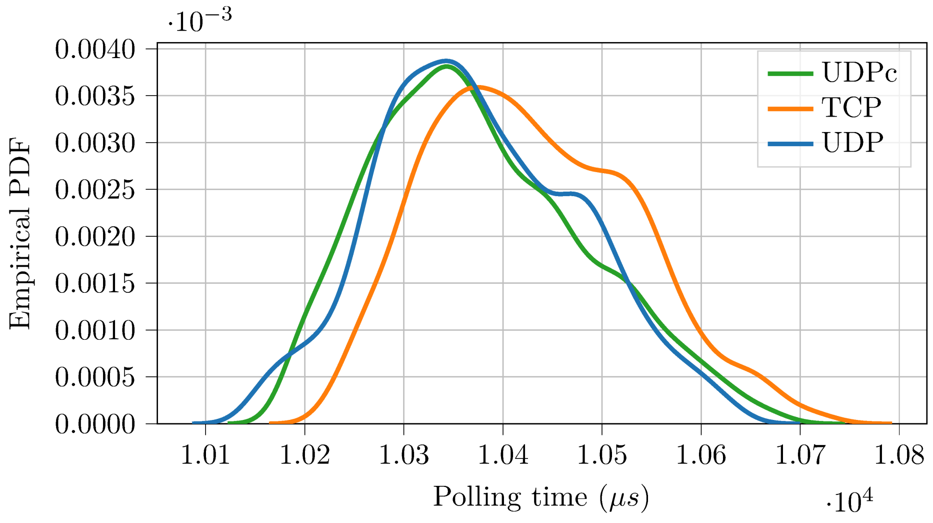


Figure 4.5: EPDF of the polling time for different transport layer on simulated channel with both PLR and PER 10^{-2} and constant delay of 5 ms

The adoption of TCP as transport protocol, in general, slightly worsened the behavior of the polling time. Indeed, mean, minimum, and maximum are higher concerning the correspondent cases in which both variations of UDP are used. This is also confirmed by Figure 4.5, which reports a comparison of the EPDFs. As can be seen, although the trend is very similar, the use of TCP leads to a slightly wider EPDF which is likely due to the greater complexity of the TCP protocol with respect to UDP. Indeed, the TCP protocol adds a set of messages for the flow control and re-transmissions that, on the one hand, has the beneficial effect of improving communication reliability, while, on the other hand, it increases both the polling time and its randomness. Moreover, compared with UDP, TCP has a more extended header, and the execution of its protocol stack requires a longer time since it is more complex, with the consequent negative impact on the polling time.

At first glance, the beneficial effect of introducing UDPc appears clear. Indeed, as shown in Table 4.2, all the statistics, except for the standard deviation, decrease with respect to TCP, obtaining performances very similar to those of standard UDP. The behavior is also confirmed by Figure 4.5, in which the EPDFs of UDP and UDPc are practically overlapping. The slight increase of the maximum and minimum polling time is most likely due to the additional operations (conditional branches, store, and comparison of memory areas) that the caching layer has to perform compared to classic UDP. Moreover, contrarily to TCP and UDP, the caching layer is executed in the userspace of the operating system in which the process can be preempted in favor of other tasks of the kernel or userspace. This mechanism involves an unavoidable overhead, thus explaining the considerable increase in the standard deviation.

4.4.2 IEEE802.11 Link configuration

In this section, the packet loss and polling time of the three considered transport layers are assessed on the same experimental setup proposed in section 3.4.1. A first insight is provided by Table 4.3, which reports the total exchanged packet and the correspondent packet loss. Following the experiments presented previously, also in this case, there is confirmation that UDP is not suitable for the transfer of SPDUs due to the not negligible packet loss. Contrary to the simulated channel experiment, TCP presents a single packet loss (one lost over nearly 8 million exchange frames). However, it is interesting to note that packets marked as lost in TCP, were slave response frames actually arrived, but after the watchdog timeout expired (set to 250 ms) between receiving two subsequent SPDUs. Regarding UDPc, the results confirm what has been observed with the simulated channel, i.e., there is no packet loss, thus confirming the effectiveness of the caching layer between UDP and FSoE. These results also allow drawing some interesting considerations about the MAC/PHY layers. It appears evident that the retransmissions performed by the 802.11 layer are insufficient to ensure the successful delivery of the packets. Indeed, by adding retransmission at UDP level, the problem is solved. As a result, theoretically, increasing the 802.11 retry limit would have the same effect as UDPc. However, such modifications would require the use of custom modified versions of the 802.11 layer, which are often difficult to implement or obtain as in the Raspberry pi used in these experimental sessions. Nonetheless, such a possibility will be investigated in future work.

	Packet loss		
	Total pkt	pkt lost	% pkt lost
UDP	7817020	25	0.00032
TCP	7457630	1	0
UDPc	7726440	0	0

Table 4.3: Lost FSoE Packet Statistics for different transport layer on IEEE802.11

A final insight is given by Figure 4.6 [2], which reports the EPDF of the polling time with the IEEE802.11 link, while Table 4.4 reports its detailed statistics. Once again, the result confirms what was observed in the previous experimental sessions. As can be seen, with UDP, it is possible to obtain the lowest polling time, followed by UDPc and finally

[2]To ease the readability of the plots in Figure 4.6 report data comprised in the interval median ± two standard deviation.

by TCP. The effects of the introduction of the real communication channel appear evident. In fact, there is a considerable increase in the standard deviation and in the maximum polling time, although in the case with the simulated channel, this was set to introduce a considerably higher transmission delay (10 ms). This is due to the actual time to transmit the frame in the physical medium and the possible retransmissions (interleaved by random backoff times) that could be necessary to deliver a packet successfully. On the other hand, contrary to the simulated channel, the standard deviations of both implementations that use UDP are quite similar likely as an effect of the randomness in accessing the physical medium introduced by Wi-Fi that masks those introduced by the operating system.

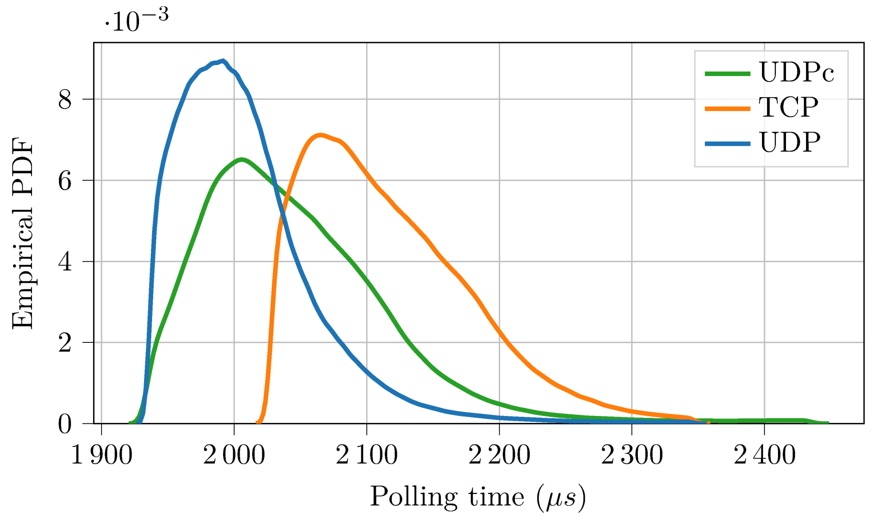


Figure 4.6: EPDF of the polling time for different transport layer on IEEE802.11

Polling time (µs)				
	mean	std	min	max
UDP	2277.18	1113.25	1183.89	43128.30
TCP	2366.40	1144.27	1257.90	225480.00
UDPc	2281.71	1113.14	1187.54	51522.90

Table 4.4: Polling time statistics for different transport layer on IEEE802.11

4.5 Discussion

First of all, it is worth remarking that in the proposed implementations, neither the protocol stack nor the FSoE stack has been modified concerning what is described in the relative standards. This is the first key result as the adoption of FSoE over any of the proposed transport protocols does not require the modification of the stack itself and therefore the designed SIL has been maintained. In this way, it is possible to use safety stacks that have been certified out-of-context (i.e., regardless of the application for which they are used) without needing a new safety assessment. Adopting UDPc does not lead to any UDP modification since it is an additional software component that acts as an interface between the safety and transport layers. Furthermore, the caching layer is both protocol and platform agnostic, i.e., its operation does not depend on the hardware/software platform in which it is used, nor on the protocols used for the PDU exchange. This makes it highly interoperable and can be used in any embedded platform.

Another key result of the experimental session is that the adoption of TCP and UDPc has dramatically decreased lost packets. Indeed, as can be observed in the detailed comparison reported in Table 4.3, with TCP, in the worst case, only one packet was lost, whereas, in the same operational conditions with UDP, the lost packets were 25. It has to remark that the single packet marked as lost in TCP is a response frame from the slave that actually arrived but after the expiration of the watchdog timeout T_W. The experiments also confirm the effectiveness of the adoption of TCP and UDPc with the simulated channel in which the impairments were set to emulate a high error–prone transmission channel, probably far beyond even the worst working conditions that can occur in an industrial plant. Both TCP and UDPc allowed to successfully exchange SPDUs without packet losses and without resetting the FSoE stack. This is a significant result since it allows to state that, with both TCP and UDPc, the reliability of FSoE over Wi-Fi may reach a level similar to that of a wired system like EtherCAT, even if at the expense of the timeliness of the protocol. Thus, it is envisaged that with the adoption of these transport protocols in the black channel, safety applications certified up to SIL3 can be implemented even over wireless connections. Regarding the timeliness, in both experimental configurations, it has been observed that the overhead introduced by TCP is such as to imply a considerable increase in polling time. On the other hand, UDPc maintains the fundamental characteristics of UDP. It allows obtaining superior performance (in the order of hundreds of microseconds) to TCP while ensuring the same degree of reliability in the transfer of SPDUs. As a general observation, it can be said that the safety over wireless systems may not satisfy the requirements in terms of SFRT precisely because of the longer transfer times of the SPDUs and the inability to guarantee tight deadlines compared to the wired counterpart. This situation can be mitigated by adopting TSN on WiFi which will allow achieving wire-equivalent reliability with deterministic time and timeliness performance over wireless (Cavalcanti, Perez-Ramirez, Rashid, Fang, Galeev, and Stanton (2019)). A final consideration can be made about the effectiveness of the black channel approach. It has been demonstrated that a safety protocol designed with this method can be implemented on networks different from that for which it has been conceived. However, it has been evidenced that not all networks can provide satisfying performance. Indeed, as we have seen, the number of lost packets when UDP over Wi-Fi is used, particularly with high PER and PLR, does not allow safety applications to reach SIL3 because the number of FSoE connection re-initializations is too high to ensure a residual error rate less than $10^{-9}/h$.

5

Tuning of a simulation model for the assessment of Functional Safety over Wi-Fi

Moving from wired to wireless safety communication is becoming a hot and challenging research topic in the industrial context, as well as in other safety critical fields, such as automotive (Xie, Li, Han, Xie, Zeng, and Li (2020)) and avionics (Allouch, Koubaa, Khalgui, and Abbes (2019)). Wireless safety solutions can be derived, for example, by exploiting the ones designed with the black channel approach described in IEC 61784-3. In this chapter, we address the widespread Fail Safe over EtherCAT (FSoE) (IEC 61784-3-12) functional safety protocol. In particular, moving from a prototype implementation of the FSoE protocol over a Wi-Fi network (Morato et al. (2019)), we developed a realistic simulation model based on Objective Modular Network Testbed in C++ (OMNeT++). The model has been carefully calibrated thanks to the results obtained from the prototype so that it can be used to simulate more complex configurations. In this respect, an example is reported where a WiFi-based FSoE network comprising a master and five mobile/fixed slaves is simulated.

5.1 OMNeT++ Simulation Model

This study aims to implement an accurate simulation model, able to reproduce realistic representations of the industrial wireless environment behavior, to provide a tool for the performance assessment of wide wireless industrial safety networks. To this purpose, the FSoE over Wi-Fi protocol has been implemented using the widespread, discrete event OMNeT++ simulator. In particular, OMNeT++ is widely adopted to simulate communication networks and model the surrounding electromagnetic environment, allowing to build protocols exploiting existing modules. In our case, this feature is particularly useful since we implemented FSoE using UDP and the validated Wi-Fi stack made available by OMNeT++ (Bredel and Bergner (2009)).

However, these models often implement generic calibrations that can simulate a wide range of scenarios but are unlikely to reproduce specific use cases. Therefore, to carefully simulate a Wi-Fi-based FSoE network, it becomes imperative to set up a precise calibration phase of both the channel errors models and the polling time, which is representative of the time necessary to complete the communication cycle between two devices.

5.2 Calibration of the channel error model

To the aim of the IEEE802.11g OFDM error model calibration, we have at first determined the PER-SNR relationship through experimental measurements on the channel. We then exploited a feature of the OMNeT++ framework, which allows as to feed the simulator with suitable lookup tables representing the PER-SNR relationship. In this way, we directly employed data from the field to reproduce a very accurate calibration of the channel model within the simulator.

To this extent, we have reproduced the experimental setup proposed by Tramarin et al. (2016) to obtain measurements of the PER-SNR function. With an approach similar to Peserico, Fedullo, Morato, Tramarin, Rovati, and Vitturi (2021a), we fine tuned the main parameters of the OMNeT++ Wi-Fi channel model. For repeatability purposes, several tests for each specific SNR value have been executed in the simulation and experimental setups, and the corresponding PER was evaluated. The results are shown in Figure 5.1, where the PER-SNR curves obtained with OMNeT++ are compared with the experimental ones. Table 5.1 reports the number of lost packets corresponding to the reset of the FSoE connection. As can be seen, the values are relatively similar, thus demonstrating the quality of the calibration of the error model.

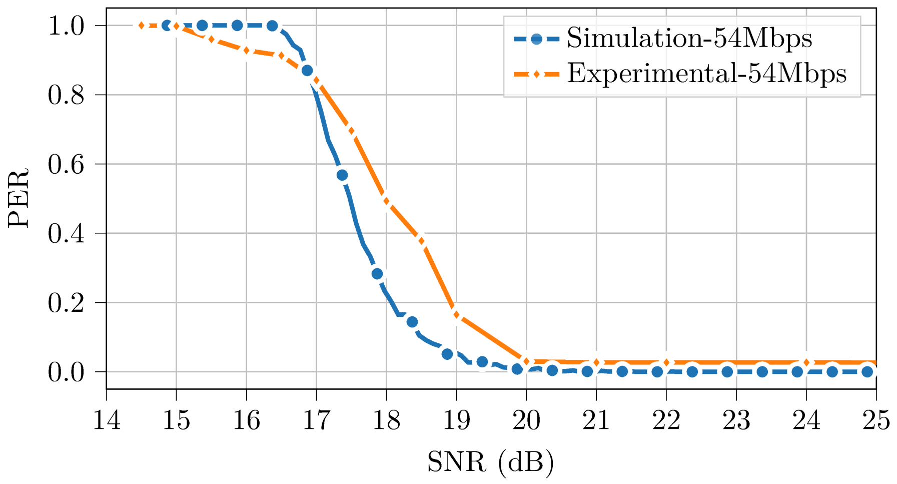


Figure 5.1: Experimental and simulated PER-SNR curves after calibration.

5.3 Calibration of the polling time

We subsequently focused on the FSoE prototype implementation, specifically evaluating the Polling Time (T_p) between an FSoE Master and the FSoE Slave. T_p is defined, in this context, as the time elapsed from the generation of an FSoE frame transmission request by the master and the reception of the confirm primitive from the slave. Morato et al. (2019) pointed out that it includes the time necessary to execute both the FSoE and the underlying protocol stacks in both master and slave, and the time to transmit the safety frame and the slave's answer message. The latter time, assuming no collision during the transmission (Tramarin et al. (2016)) may be considered deterministic. Conversely, the former time usually depends on the device's characteristics where protocols and applications are implemented, such as the computational capabilities and operating system calls, memory management, etc., which may introduce random latencies and jitter. OMNeT++, in some way, tries to take into account these uncertainties by introducing some uniform random delays, but obviously, they cannot correspond to real use cases.

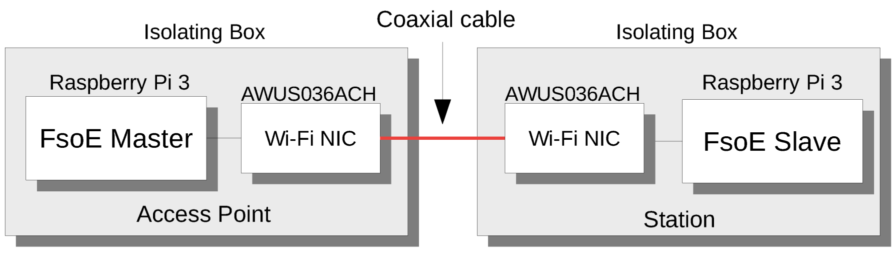


Figure 5.2: Experimental set-up

Therefore, a different set of tests were relevant to the calibration of the simulator with respect to experimental values. To this goal, a suitable measurement setup has been designed, as shown in Figure 5.2.

All the experiments have been carried out on Raspberry Pi Model 3B boards which run a general-purpose operating system Raspbian OS (Kernel version 5.4.83). Each device has been equipped with an external Alfa AWUS036ACH USB Wireless Network Interface Controller (WNIC) set to operate in the 2.4 GHz band with IEEE802.11g modulation standard and output power of 30 dBm. Moreover, the rate adaptation features have been disabled, thus using a fixed 54 MBps rate. Tests have been conducted in an industrial area, whereby it is necessary to minimize as far as possible the influence of external factors, for example, other Wi–Fi stations or background noise. For this reason, the WNICs antennas have been connected via a coaxial cable, thus simulating an ideal transmission medium. To further increase the robustness to external noise, the Raspberry Pis and the WNICs have been embedded into separate shielding boxes. Finally, a variable RF attenuator, with an attenuation range of 50-110 dBm, has been inserted in the coaxial transmission line to properly control the real transmission medium's attenuation.

The FSoE Master and FSoE Slave have been implemented on two different Raspberry Pi boards, respectively configured as Wi-Fi Access Point (AP) and Station (STA). The communication between the Master and the Slave is hence managed without intermediate devices. The simplest FSoE SPDU (7 bytes) has been encapsulated into a UDP frame to carry safety data. It is worth noting that the FSoE stack runs on the Raspberry Pi as normal non-prioritized processes. Moreover, SPDUs are sent continuously in a non-scheduled way. In particular, after receiving the safety frame SF_i from the master, the slave answers with the frame AF_i. Then, the master sends a new safety frame SF_{i+1} immediately after receiving the answer AF_i from the slave. In each device, a watchdog of 250 ms monitors the FSoE connection cycle to detect possible delays on the network. If a device does not receive any answer from the communication partner within the specified timeout, the frame is marked as lost, and the FSoE connection is re-initialized.

Several experimental sessions have been conducted to test the medium with different attenuation values that have been varied in 1dB steps. Correspondingly, for each measurement session, acquisition of more than 50000 unique values of the polling time T_p has been analyzed.

We used the probability densities obtained from the experimental measurements to calibrate the polling time model. Each of these was obtained for different channel attenuation values. Using the Inverse Transform Sampling method, the delays to be used in the transmission are sampled from the experimental densities according to the received signal strength. In practice, when the Master issues the transmission of an SPDU, the simulator calculates what will be the channel attenuation and, therefore, the power of the received signal. Based on this, the simulator chooses the density from which to sample, and schedules the response message from the slave after the delay generated by the sampling. The outcome of the polling time calibration is shown in Figure 5.3, while a more detailed statistic is reported in Table 5.1. As shown in Fig. 5.3, the trend of the mean, minimum and maximum values of the polling time is rather similar for both the experimental and simulated sessions.

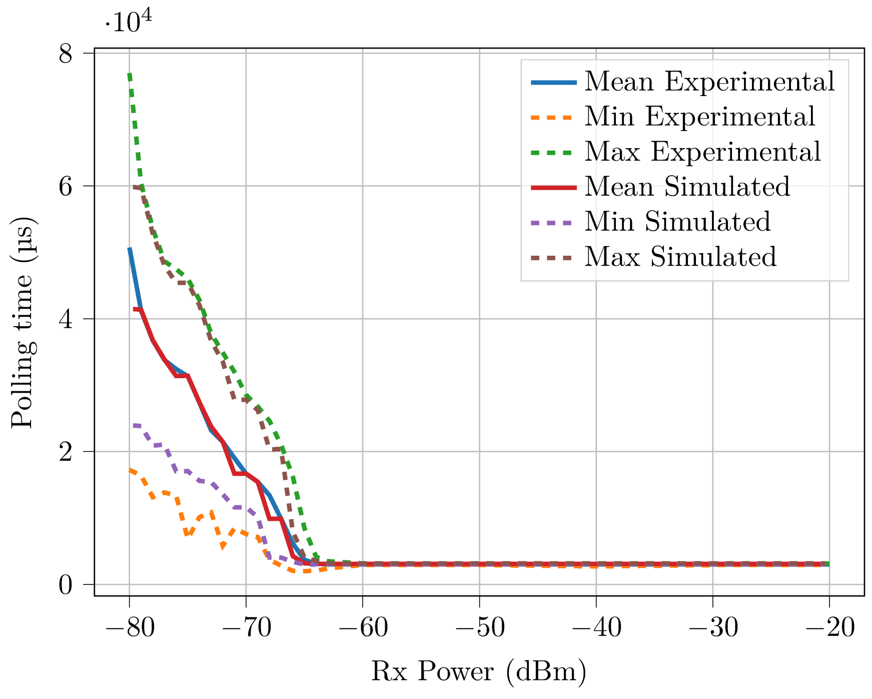


Figure 5.3: Comparison of mean, maximum and minimum polling time on the experimental and simulated setup

Indeed, for high reception powers, the trends are practically identical. For powers lower than -60 dBm, the polling time values follow the same trend but with a higher error. This observation is also confirmed by Figure 5.4, which shows the Mean Square Error (MSE) calculated on the average polling time.

As can be seen, the error is almost zero for relatively high receiving powers, while it tends to increase as the intensity of the received signal decreases. This phenomenon is due to the model of path loss used in the simulator which is the *LogNormalShadowing*. The path loss model simulates the attenuation of the signal and calculates the probability that the signal can be received. It also adds a certain degree of variability to the signal

	rxPower (dBm)	Polling time (µs)				PER	MSE on mean (%)
		Mean	Std	Min	Max		
Experimental							
	-80.0	50720.27	11510.14	17271.40	76988.70	0.030	-
	-76.0	32437.22	6756.42	13383.80	47484.70	0	-
	-73.0	23225.14	6050.65	10836.40	37790.80	0	-
	-70.0	16663.57	4124.12	7597.51	28525.20	0	-
	-67.0	9870.57	3460.87	2855.14	21021.10	0	-
	-64.0	3116.34	220.08	2076.29	4144.56	0	-
	-50.0	3010.49	23.01	2931.70	3139.15	0	-
	-20.0	3010.20	21.32	2930.30	3130.82	0	-
Simulated							
	-80.0	41473.87	8602.50	23940.54	59851.54	0.027	18.23
	-76.0	31377.75	6506.41	17041.85	45395.85	0	3.26
	-73.0	21427.64	4985.96	13059.43	34340.43	0	7.73
	-70.0	16684.58	3954.34	11635.10	27807.10	0	0.12
	-67.0	9877.29	3260.08	4049.86	20378.86	0	0.06
	-64.0	3106.29	82.38	3035.66	3535.66	0	0.32
	-50.0	3084.09	21.70	3014.20	3173.20	0	2.44
	-20.0	3085.52	19.16	3048.02	3170.02	0	2.50

Table 5.1: Statistics of the polling time and PER

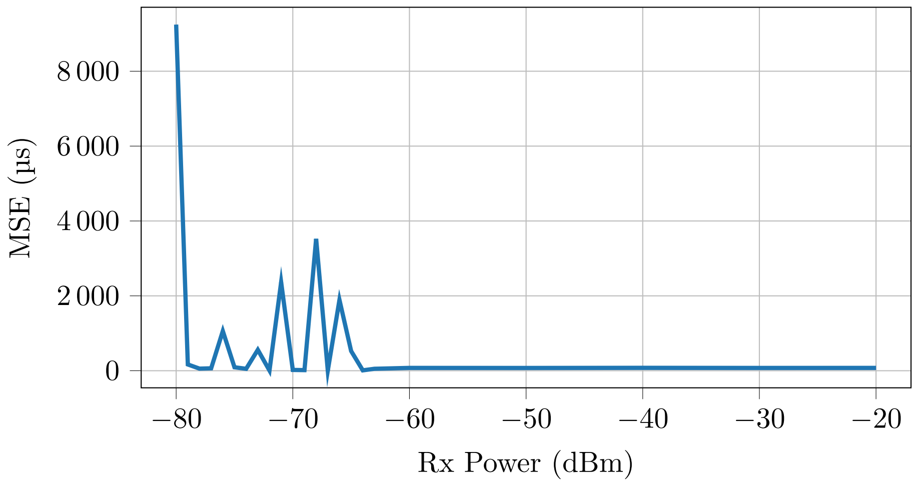


Figure 5.4: MSE

reception time to simulate the attenuation effects.

Therefore, the uncertainties introduced by the path loss model overlap with the delays introduced by the polling time model, causing it to deviate slightly from the experimental trend. The solution to this problem will require the implementation of a completely custom path loss model, which will be left for future work. In the current state of the simulator, the path loss model has been calibrated to minimize these superimposition effects.

In general, the accuracy of the calibration can be confirmed by the MSE, which, except for some isolated cases, remains less than 4%. This is also confirmed by Figure 5.5, which compares the probability density function of the polling time. As can be seen, they are definitely very similar.

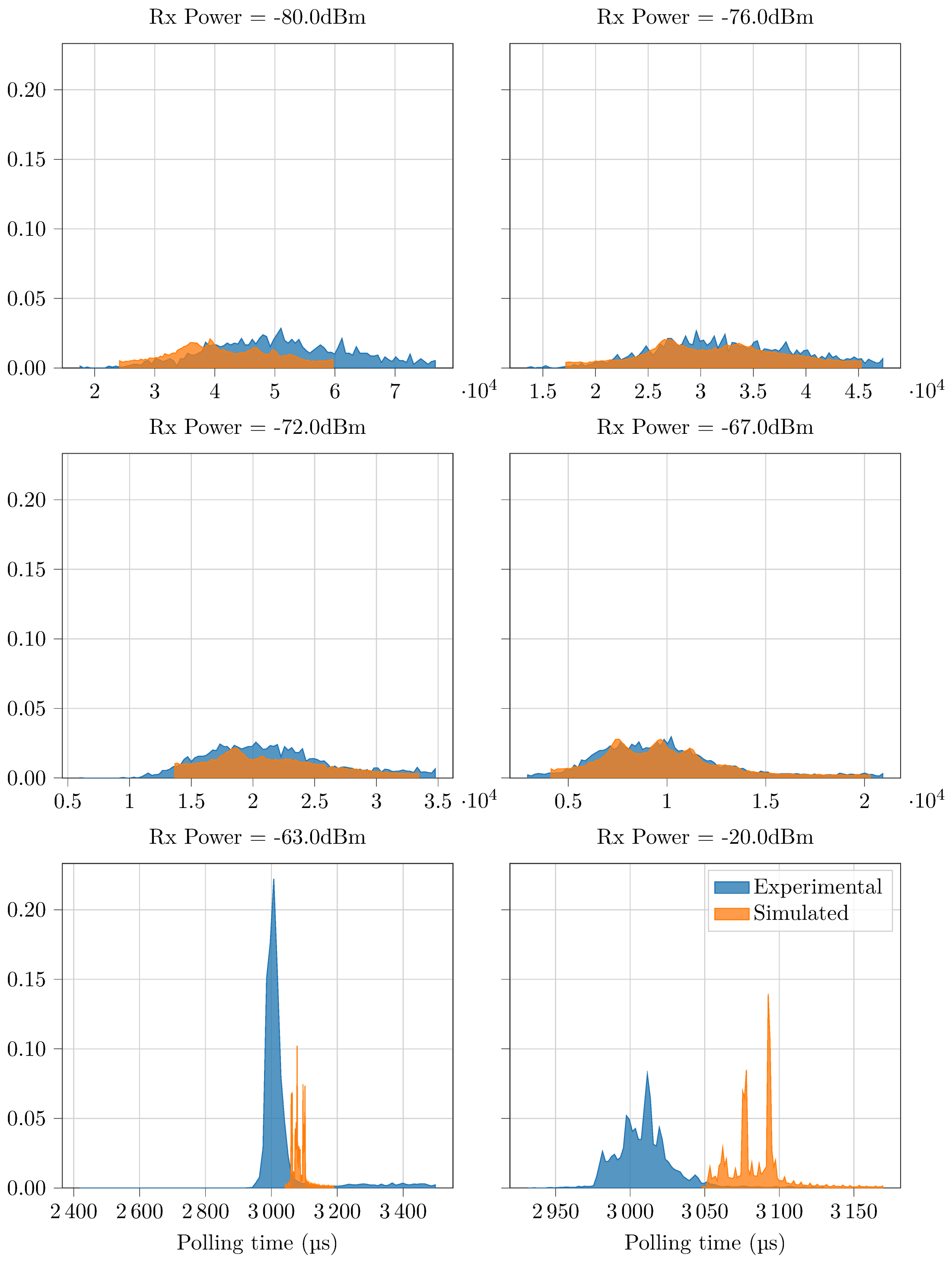


Figure 5.5: Comparison of the probability density function of the polling time in the experimental and simulated setup

5.4 Simulation with multiple slaves

The simulation of a realistic multi-node network needs particular care and requires the execution of several tests. In this section, we analyze the outcomes of some simulation sessions we have carried out towards assessing the proposed simulation model. We refer to the prototype network described in Figure 5.6, composed of one FSoE master and five FSoE slaves. The position of the nodes with respect to each other has been carefully selected to reproduce and test three different communication scenarios. Firstly, aiming at analyzing the protocol behavior in the absence of mutual interference, Slaves 1 and 2 are placed relatively distant to each other, thus reproducing an ideal situation. On the contrary, slaves 3 and 4 have been placed close together to study how possible interference can affect the polling time and the PER. Finally, slave 5 has been placed far from the master to analyze the impact of a great attenuation, i.e., a relatively low SNR, on the polling time and the PER.

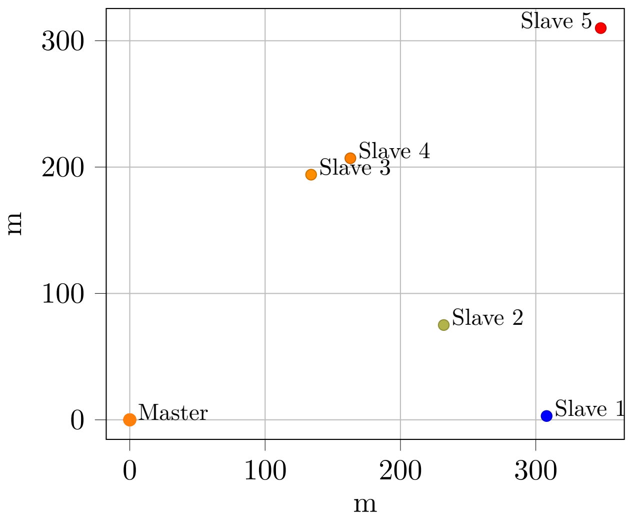


Figure 5.6: Positions of the nodes

Simulation statistics of both the polling time and exchanged packets for each node are reported in Table 5.2.

Comparing the behavior of Slaves 1 and 2, it is possible to underline that the latter introduces a slightly lower polling time, while both do not experience any packet loss. This is reasonable as they do not have other nodes nearby, but the distance between Slave 1 and the Master is higher than the Slave 2 one. Conversely, Slaves 3 and 4 introduce mutual interference, as can be noted from both the polling time (T_p) and the Packet Error Rate (PER). Indeed, the polling times of Slaves 3 and 4 are quite similar to those of Slaves 1 and 2, although the latter ones have a greater distance from the master.

Slave	**Polling time (µs)**				**Packets**			
	Mean	Std	Min	Max	Sent	Acknowledged	Lost	PER
1	33932.20	6581.77	21070.06	48071.06	71997	71997	0	0.000000
2	27618.30	6540.25	15593.58	42273.58	71997	71997	0	0.000000
3	27390.94	6506.25	15576.63	42060.63	71999	71999	0	0.000000
4	31431.96	6545.19	17087.76	45433.76	72000	71998	2	0.000028
5	105864.38	52708.85	28557.11	239921.11	71998	52683	19315	0.268271

Table 5.2: Statistics of the polling time in simulation with multiple slaves

Furthermore, Slave 4 also experiences some packet loss. Finally, Slave 5 introduces higher T_p and PER, thus underlying the impact of the attenuation on the communication.

6 Evaluation of the Safety Function Response Time in a Functional Safety over Wi-Fi Network

Recently, industrial automation and distributed measurement systems have seen an increase in the use of wireless communication networks, which is leading to an improvement in scalability, efficiency, flexibility, and cost reduction (Wollschlaeger, Sauter, and Jasperneite (2017)). The fundamental aspect in the use of wireless in industrial fields is the opening towards new categories of autonomous devices that can actively collaborate with human personnel in the production process. However, the safety implications of this new paradigm are very strong. In traditional plants, hazardous machineries are often protected by physical barriers that protect the operating space. The development of safety fieldbuses and safety strategies for networked industrial systems (Vitturi et al. (2018)) has further improved safety performance by making it possible to create devices that operate in a coordinated manner and are able to guarantee a very high degree of safety (for example SIL3 with residual dangerous fault probability $< 10^{-9}/h$). In mobile and collaborative devices, the use of physical barriers is clearly not a viable solution, and the coordination between sensors, actuators, and control systems must be as reliable as possible. This becomes possible by ensuring the correct transfer of safety-critical data, and therefore, the development of the same safety communication techniques present in wired communications becomes of fundamental importance. The topic has been the subject of numerous researches in which various safety protocols conceived for wired networks have been successfully ported to wireless networks using the black channel approach. For example, Åkerberg et al. (2011) implemented ProfiSafe on WirelessHART, presenting a proof-of-concept experimental setup in which the achievable performance in terms of cycle time was analyzed. Another application was proposed by Hadz, iaganovic, Atiq, Blazek, Bernhard, and Springer, where OpenSafety was implemented on Message Queue Telemetry Transport (MQTT) to implement a Plug&Play distributed safety network. In chapters 3 and 4, we proposed the implementation of FSoE on IEEE802.11 WLAN, analyzing in depth the performance in terms of polling time T_p (that is the time necessary for a complete cycle of communication between Master and Slave) and of packet loss. The impact of different transport protocols on the T_p was also analyzed. In the context of safety communications, the analysis of communication time takes on a fundamental aspect. Indeed, although the IEEE802.11 WLAN demonstrated its effectiveness in the industrial communication scenario, in that it is able to ensure fast message delivery times, with jitter and reliability acceptable by diverse types of applications (Tramarin, Mok, and Han (2019)), their performance figures are not yet comparable with their RTE counterparts. Generally speaking, in Networked Control Systems (NCSs), jitter and long communication times may compromise the control system's responsiveness (i.e., the reaction time). It also affects a very critical aspect in safety critical communications: the time between the detection of an error and the system's transition into a safe state, which is defined as Safety Function Response Time (SFRT). The SFRT is defined in the IEC 61784-3-3 (ProfiSafe) standard as a metric for safety-critical automation systems, which defines the worst case time to reach the safe state of the system in the presence of errors or failure in the safety function or in the communication medium itself. Although SFRT is explicitly defined for ProfiSafe (primarily conceived to work with wired networks), its definition can be carried over to other safety communication protocols and wireless networks. Indeed, Åkerberg et al. (2011) have proposed an approach for estimating SFRT in SISO systems using IEEE 802.15.4 based wireless networks. Furthermore, Pimentel and Nickerson (2014) have extended the work of the previous authors, studying the applicability of SFRT to MIMO systems. In this chapter, we present a framework for estimating SFRT in SIMO systems using FSoE over Wi–Fi. The estimates will be verified using the simulator proposed in chapter 5. Finally, the evaluation of SFRT will be presented by simulating a real industrial environment composed of autonomous mobile robots.

6.1 Background

The Safety Function Response Time (SFRT) is defined in the IEC 61784-3-3 for SISO systems as

$$SFRT = \sum_{i=1}^{n} WCDT_i + \max_{i=1,2,\ldots,n} (WDTime_i - WCDT_i) \tag{6.1}$$

$$= TWCDT + \Delta T_{WD} \tag{6.2}$$

where:

- n is the number of entities on the network.
- $WCDT_i$ is the worst case delay time of entity i.
- $WDTime_i$ is the watchdog timer of entity i.

Let consider Figure 6.1, which is a representation of Eq. 6.1.

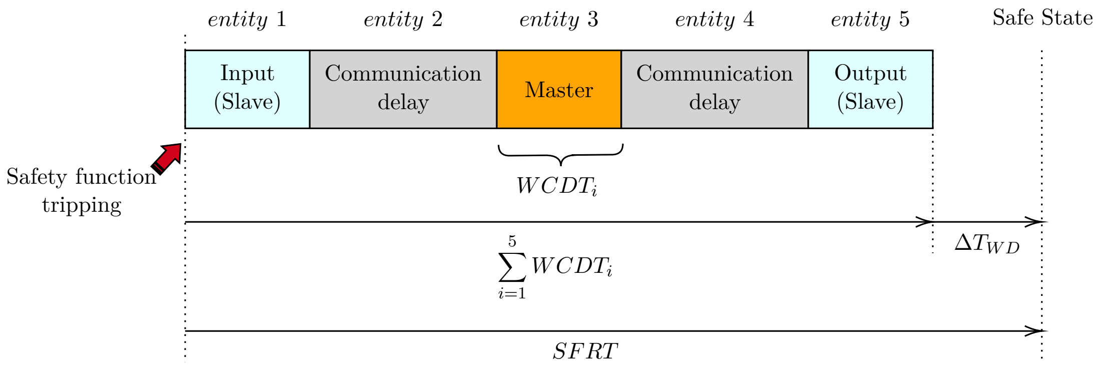


Figure 6.1: Representation of the SFRT for SISO systems as defined by IEC 61784-3-3

As can be seen, an entity is meant as the single component that is part of the chain between the input that triggers the execution of the safety function and the output that is forced to the safe state. Each entity has its own execution time $WCDT_i$. For example, entities 1, 5, and 3 respectively represent the processing time of the slaves and the master , while entities 2 and 4 represent the transmission delay of the channel. Therefore, the total delay $TWCDT$ is given by the sum of the delays given by the single components. In the calculation of the SFRT, the IEC 61784-3-3 considers the case in which one of the communication paths between the entities may fail. Suppose the $entity_i$ is affected by a communication failure. In this case, the $WCTD_i$ delay introduced by the $entity_i$ is equal to $WDTime_i$. Assuming that each entity has its own watchdog time, the term

ΔT_{WD} takes into account the maximum delay introduced from the $entity_i$ in case there is an error in the communication path. In other words, let be

$$j : \max_{i=1,2,\ldots,n}(WDTime_i - WCDT_i) = WDTime_j - WCDT_j \tag{6.3}$$

then, Eq. 6.1 can be rewritten as

$$SFRT = \sum_{\substack{i=1 \\ i \neq j}}^{n} WCDT_i + WDTime_j \tag{6.4}$$

6.2 Safety function response time on a FSoE over Wi–Fi network

The SFRT equations formulated by IEC 61784-3-3 are conceived for systems with a single input and single output. In more complex automation systems, interactions are hardly limited to point-to-point connections. For this reason, Pimentel and Nickerson (2014) have proposed an extension of the SFRT equation for MIMO systems. In this section, we will use a modified version of the latter equations to estimate the SFRT in an FSoE over Wi–Fi system.

Pimentel and Nickerson (2014) have extended the SFRT equation for MIMO systems, considering the partition $E = \{Input, Master, Output, TrasmissionDelay\}$ and redefining Eq. 6.1 as

$$SFRT = \sum_{i \in E}^{n} WCDT_i + \max_{i \in E}(WDTime_i - WCDT_i) \tag{6.5}$$

where $WCDT_i$ is the worst case delay time of all entities of type i, thus, representing the total worst case delay time of all the entities in the network, while $WDTime_i$ is the watchdog timer of all entities of type i. In a FSoE over Wi–Fi network, usually the trigger of a safety function originated in one of the slaves then, it is propagated to the Master and subsequently to all the other slaves in the network. In this case, Eq. 6.5 can be rewritten considering a new partition $F = \{I, O\}$, $|F| = n$ where I group all the entities from the Slave that have originated the safety function trigger to the Master, while O groups all the entities from the Master to the Output Slaves. Figure 6.2 gives the representation of the partitioning.

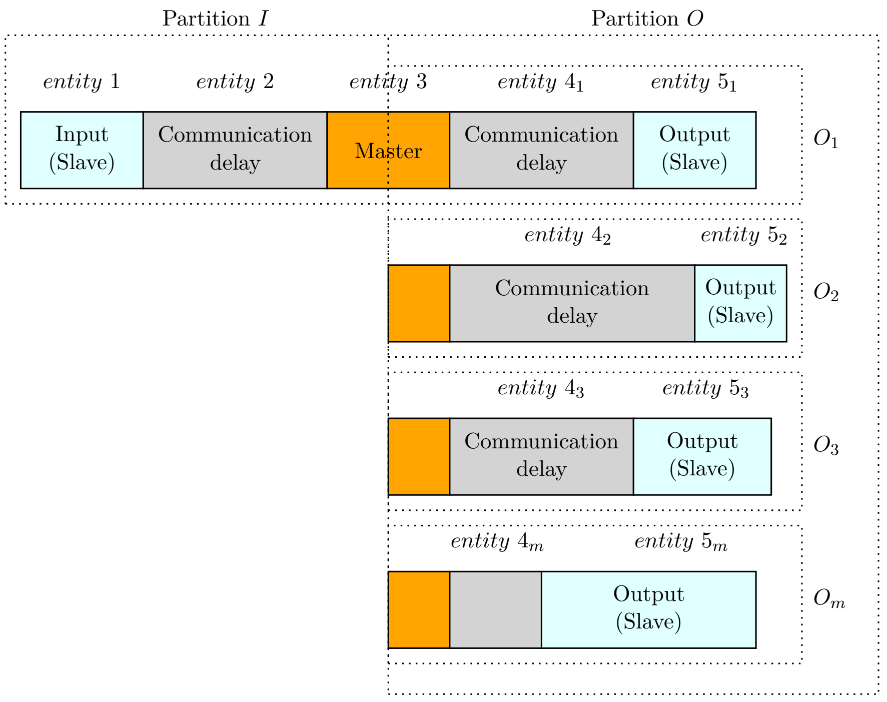


Figure 6.2: Partitioning of network entities in SIMO systems

Given the new partitioning, Eq. 6.5 can be rewritten as

$$SFRT = \sum_{i \in I} WCDT_i + \sum_{j \in O} WCDT_j + \max_{l \in F}(WDTime_l - WCDT_l) \tag{6.6}$$

Eq. 6.6 can be further redefined by considering the partitioning of O as $O = \{O_1, O_2, \ldots, O_m\}$ where O_i represent the entities from the master and a single Slave and m is the number of output Slaves, so that $F = \{I, O_1, O_2, \ldots, O_m\}$, $|F| = n$. Eq. 6.6 become

$$SFRT = \sum_{i \in I} WCDT_i + \sum_{k=1}^{m} \sum_{j \in O_k} WCDT_j + \max_{l \in F}(WDTime_l - WCDT_l) \tag{6.7}$$

About the equation Eq. 6.7 some remarks are necessary. First, the processing time of the master is considered n times as it has a separate instance for each slave. Secondly, the SFRT is calculated assuming that the polling occurs sequentially, that is, the master begins a communication with the slave belonging to the O_i partition only after the polling of the slave belonging to the O_{i-1} partition has finished. From this point of view, note that 6.2 is not exhaustive as it represents the polling of the slaves in parallel mode. Given this last observation, it can be said that the SFRT also depends on the access mode to the slaves.

In the case of parallel polling, the SFRT takes the form

$$SFRT_P = \sum_{i \in I} WCDT_i + \max_{k=1 \ldots m} \sum_{j \in O_k} WCDT_j + \max_{l \in F}(WDTime_l - WCDT_l) \tag{6.8}$$

where

$$I : \sum_{i \in I} WCDT_i = \max_{Q \in F}(\sum_{q \in Q} WCDT_q)$$

which leads to $SFRT_P \leq SFRT$. Indeed, as pointed out by Åkerberg et al. (2011), the SFRT can be enhanced by using different polling strategies.

Now, let us evaluate $\sum WCDT$ in the case of a network using FSoE over Wi–Fi. As specified in the 2.3.2 section, FSoE is a polling-based protocol, i.e., the communication cycle is initiated by the master, which waits for the response from the slave before generating a new SPDU. The slave itself cannot initialize a safety communication.

To evaluate $\sum WCDT$, let us recall the definition of Polling Time T_p: T_p is defined as the time elapsed from the generation of an FSoE frame transmission request by the master and the reception of the confirm primitive from the slave. It includes the time necessary to execute both the FSoE and the underlying protocol stacks in both master and slave, the time to transmit the safety frame and the slave's answer message. Therefore, T_p implicitly includes all processing time of the considered partition entities.

Without loss of generality, let us consider the input partition I. If the safety function is triggered, the worst case is represented by the trigger immediately after the slave has sent an SPDU response to the master. In this case, for the master to be notified of the trigger of the safety function, it is necessary to wait for a network cycle, i.e., T_p. The same condition also applies to the O_i output partitions. Suppose the master receives the safety function trigger from the slave in the partition I immediately after initiating a communication cycle with the slave of the output partition O_i. For the slave in O_i, a time T_{p_i} is required to receive the notification of the activation of the safety function, corresponding to the polling time of the slave in that partition.

Therefore, in general, it can be said that $\sum WCDT = T_p$, hence, Eq. 6.7 and Eq. 6.8 can be respectively redefined as

$$SFRT = T_{p_I} + \sum_{k=1}^{m} T_{p_{O_k}} + \max_{l \in F}(WDTime_l - WCDT_l) \tag{6.9}$$

$$SFRT_P = T_{p_I} + \max_{k=1\ldots m} T_{p_{O_k}} + \max_{l \in F}(WDTime_l - WCDT_l) \tag{6.10}$$

with

$$I : T_{p_I} = \max_{Q \in F}(T_{p_Q})$$

where T_{p_I} is the polling time of partition I, while $T_{p_{O_k}}$ is the polling time of partition $O_k \in O$.

6.3 Evaluation of the Safety Function Response Time

To provide an evaluation of the SFRT, we considered the multi slave network proposed in Chapter 5. Table 6.1 reports the polling times measured by the simulation from which, using the maximum values, it is possible to calculate both the $SFRT$ and $SFRT_P$ using, respectively, Eq. 6.9 and Eq. 6.10.

Slave	**Polling time (µs)**			
	Mean	Std	Min	Max
1	33932.20	6581.77	21070.06	48071.06
2	27618.30	6540.25	15593.58	42273.58
3	27390.94	6506.25	15576.63	42060.63
4	31431.96	6545.19	17087.76	45433.76
5	105864.38	52708.85	28557.11	239921.11

Table 6.1: Statistics of the polling time in simulation with multiple slaves in OMNeT++

It is

$$\begin{aligned} SFRT &= 239921.11 + 177839.03 + \max_{l \in F}(WDTime_l - WCDT_l)\mu s \\ &= 417760.14 + \max_{l \in F}(WDTime_l - WCDT_l)\mu s \end{aligned}$$

$$\begin{aligned} SFRT_P &= 239921.11 + 48071.06 + \max_{l \in F}(WDTime_l - WCDT_l)\mu s \\ &= 287992.17 + \max_{l \in F}(WDTime_l - WCDT_l)\mu s \end{aligned}$$

For both equations, remain to evaluate the term $\max_{l \in F}(WDTime_l - WCDT_l)$.

Considering the watchdog timer is set to 250 ms among all master and slaves, and as pointed in Section 3.4.2, the lowest $WCDT$ is given by the elaboration time on the slaves, which is the order of thousand of µs, it is $WDTime_l >> WCDT_l$ so that

$$\max_{l \in F}(WDTime_l - WCDT_l) \approx WDTime_l$$

leading to

$$SFRT = 417760.14 + 250000\mu s = 667760.14\mu s$$

$$SFRT_P = 287992.17 + 250000\mu s = 537992.17\mu s$$

The obtained SFRT theoretical values were verified using the simulator based on OMNeT++ and the network with multiple slaves presented in Chapter 5. It is a network composed of five slaves and one master that communicate via FSoE over Wi–Fi. The SPDUs are encapsulated in a UDP frame, and communication between master and slaves occurs unscheduled. To evaluate the SFRT has been simulated the trigger of a safety function in a slave and were measured the elapsed time between the function trigger and the instant in which all the other slaves received the trigger notification, i.e., they have enabled the safe state. In this simulation scenario, the safe state activation request can be either the actual trigger of the safety function or a communication error, for example, packet loss. The watchdog time $T_w = WDTime$ has been set to 250 ms. To asses the SFRT in different conditions, were carried out 1000 simulations in which the trigger of the safety function occurs at random time instants in one of the randomly chosen slaves.

The simulation results are reported in Figure 6.3, while the detailed statistics are reported in Table 6.2. In both, are reported either global statistics and those categorized by the slave that generated the trigger.

Slave	**SFRT** (µs)			
	Mean	Std	Min	Max
1	232163.41	120021.29	62244.42	436140.13
2	246799.44	116161.35	119495.67	455973.00
3	225520.75	128919.79	74778.65	434644.44
4	258786.35	143809.41	63660.59	495917.50
5	360902.26	131576.36	94596.89	480198.55
Global	289150.30	141416.21	62244.42	495917.50

Table 6.2: Statistics of the SFRT in simulation with multiple static slaves. The slave index represent the slave which have originated the safety function trigger.

As a first observation, it can be said that the SFRT has no dependence on the slave

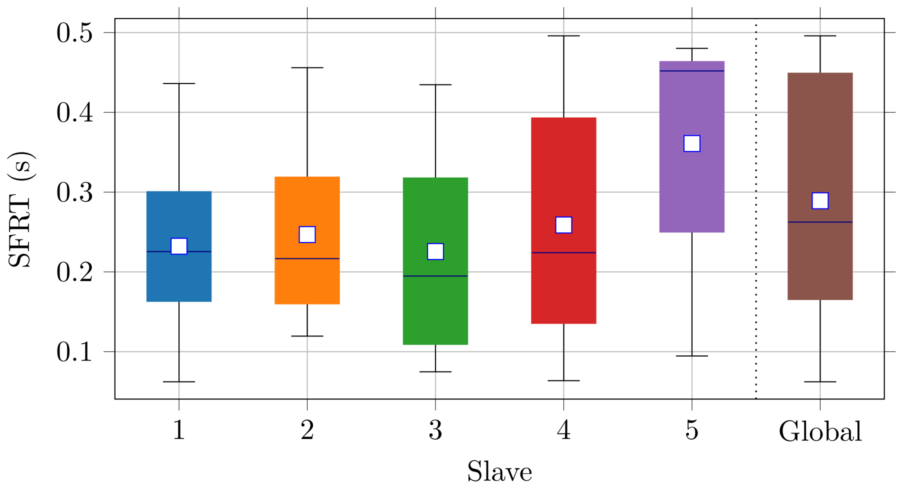


Figure 6.3: Statistics of the SFRT of the stimulated multi slave network in OMNeT++. X-axis represent the slave which have originated the safety function trigger, while y-axis report the SFRT of the network. Blue horizontal line represent the median while, white square is the mean

that generates the trigger of the safety function. As can be seen, both the minimum, average, and maximum values are very similar among all the slaves. This is a significant result since it allows to estimate, once the operating environment has been defined, a maximum value of SFRT independently of the position of the nodes. This result is also of interest for mobile nodes networks, as the global SFRT can be estimated whatever their position in the operating space.

Concerning SFRT in the worst case, as shown in Figure 6.3 and in detail in Table 6.2, the maximum value is generally slightly lower than the SFRT theoretically calculated. However, especially in slaves 4 and 5, the maximum value is close to the theoretical value, and the average value of the SFRT slightly higher than the other slaves. This is explained by the fact that both slaves have a PER higher than the other slaves, so it is more likely that a watchdog timeout condition will be generated, leading to the worst case SFRT. In any case, the actual SFRT is lower than the theoretical one, thus demonstrating that it is a conservative upper bound.

6.4 Industrial environment simulation

The need for functional safety wireless networks stems from different industrial use cases characterized by the need to have distributed devices, possibly mobile, that work in environments with very strict safety requirements. The case study presented in this section refers to a semi–automated warehouse, where mobile robots operate goods handling within protected work areas. The plant, which is schematically represented in Figure 6.4, includes two distinct workspaces. The first, highlighted in green, is the loading bay where the unloading and loading of goods from trucks by human personnel takes place manually. The second area is highlighted in red and is a fully automated area where mobile robots move goods from storage areas for further processing. For safety reasons, the automated area is forbidden to human personnel.

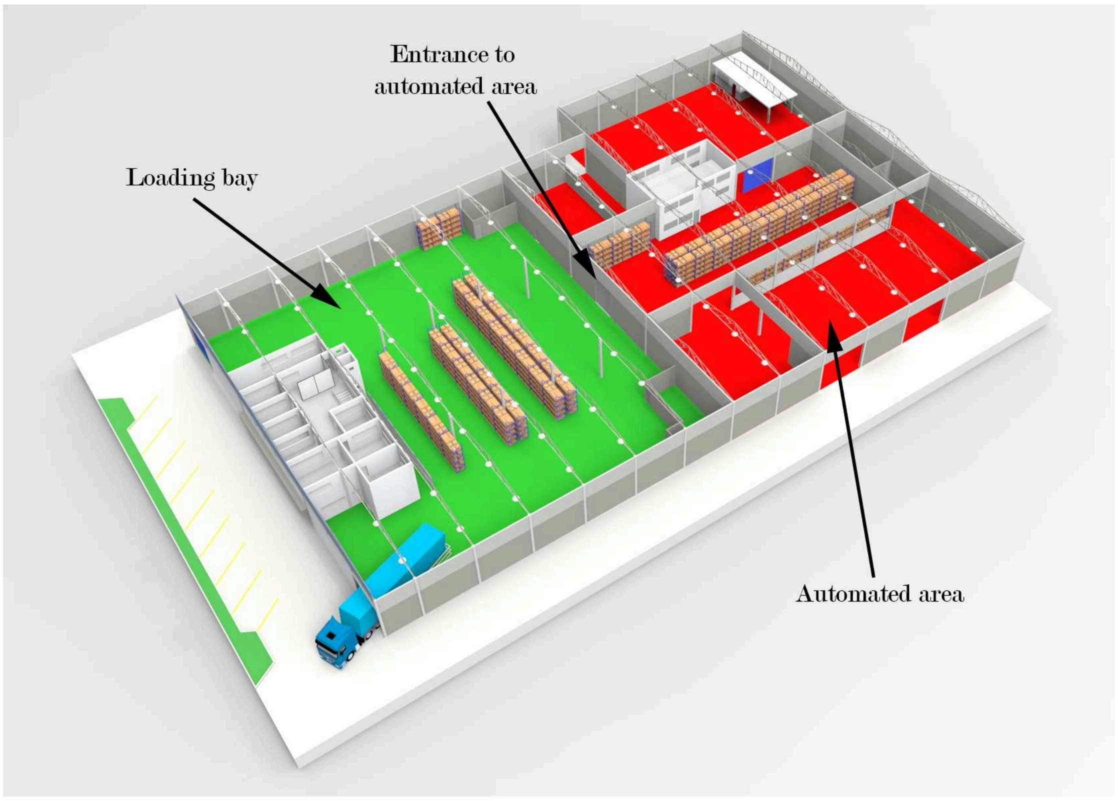


Figure 6.4: Example of semi–automated warehouse. Credit to Jesus Sobalvarro

When the goods are deposited in the warehouse (green area), they are periodically transferred manually to the automated area. The two areas are communicating through a single entrance. Four autonomous mobile robots operate in the automated area. They are connected via a safety network based on FSoE over Wi–Fi to a centralized safety master located at the center 5 m above the ground of the automated area in order to provide an optimal connection converge. The robots represent mobile safety slaves. Another safety slave, in this case, fixed, is represented by the barriers placed at the entrance to the automated area. One of the safety requirements is that when the safety barriers are opened, the Safe Torque Off (STO) (IEC 61800-5-2) safety function must be activated in the mobile robots, which consists in deactivating the power supply to the electric drives that allow the robot to move so that they are stopped in the shortest possible time.

The safety master cyclically polls all the slaves in an unscheduled way. In particular, it cyclically requests the status of the barriers placed at the entrance. If these are open, the safety master, at the first useful network cycle, sends to the mobile robots the STO safety function activation request. The latter can also be activated by loss of communication with one of the robots or by one of the robots themselves due, for example, to internal faults. However, in this use case simulation, only the trigger of the safety function due to barrier opening or communication problems were considered.

The environment represented in Figure 6.4 (red area) has been implemented in OMNeT++. The walls were modeled as concrete walls of thickness 10 cm using the attenuation model *DielectricObstacleLoss*. The power loss is determined by computing the accurate dielectric and reflection loss along the straight path considering the shape, the position, the orientation, and the material of obstructing physical objects. The robots have been modeled as point objects that move in the operative space following a random walk trajectory with a random speed within the range of 1 m/s to 1.5 m/s. It is also assumed that the robots are massless, i.e., the stop coincides with the receipt of the trigger of the safety function in the safety application layer of the slave. All in all, this is a realistic hypothesis since, if equipped with emergency brakes (IEC 61800-5-2), following the activation of STO, the robots stop in a negligible time. An example of the trajectories performed by the robots is shown in Figure 6.5.

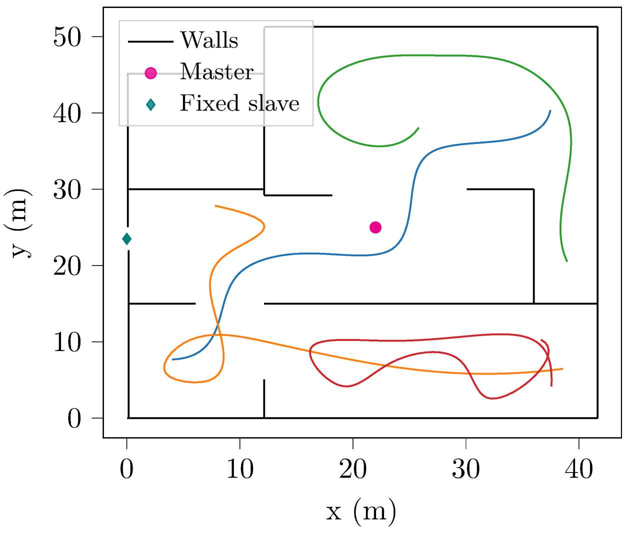


Figure 6.5: Top view of the simulated automated area in Figure 6.4 showing location of the master, fixed slave and an example of trajectories of the mobile robots

In order to evaluate the SFRT, i.e., the time elapsed between the opening of the safety barriers and the complete stop of all the robots, 1000 simulations were carried out in OMNeT++. Initial positions of the nodes and time instant in which the trigger of the safety function occurs have been chosen randomly in each simulation.

As a first result, global statistics of polling time for all 1000 simulations are reported in Table 6.3, in which slave 1 refers to the barriers, while the others refer to mobile robot. As can be seen, the statistics are very similar among all slaves. This result is quite obvious since, being the master placed in the center of the operating area, on average, the distance between the master and the slaves remains in the same range, resulting in similar polling times. In fact, as seen in Chapter 5 in the simulations with static slaves, nodes placed at a similar distance from the master, and therefore with the same channel attenuation and SNR, have comparable polling times.

Slave	**Polling time** (µs)			
	Mean	Std	Min	Max
1	4867.53	1781.71	2957.13	10828.31
2	4884.32	1795.62	2937.07	10974.59
3	4942.18	1799.11	2993.03	37152.90
4	4876.03	1789.71	2978.20	10837.58
5	4879.58	1792.74	2958.15	10779.85

Table 6.3: Statistics of the polling time in simulation with multiple mobile slaves. Slave 1 is the barrier, while the others are the mobile robots

Considering that the watchdog time $T_w = WDTime$ has been set to 250 ms, using the polling time statistics, it is possible to calculate both the $SFRT$ and $SFRT_P$ using respectively Eq. 6.9 and Eq. 6.10, whose lead to

$$SFRT = 330573.23\mu s$$

$$SFRT_P = 298127.49\mu s$$

The theoretical values have been compared with the simulation results reported in Figure 6.6, which represent the EPDF of the SFRT[1], whereas its statistics are provided in Table 6.3.

As can be seen, the results are particularly satisfactory since the safety function is activated in the mobile nodes on average in about 13 ms. This value can be considered satisfactory on a wide spectrum of applications. Indeed, in this specific use case, such

[1]To ease the readability of the plot in Figure 6.6, data greater than median + three standard deviation are omitted.

SFRT (µs)			
Mean	Std	Min	Max
13181.83	8093.87	7221.98	271952.15

Table 6.4: Statistics of the SFRT in simulation with multiple mobile slaves.

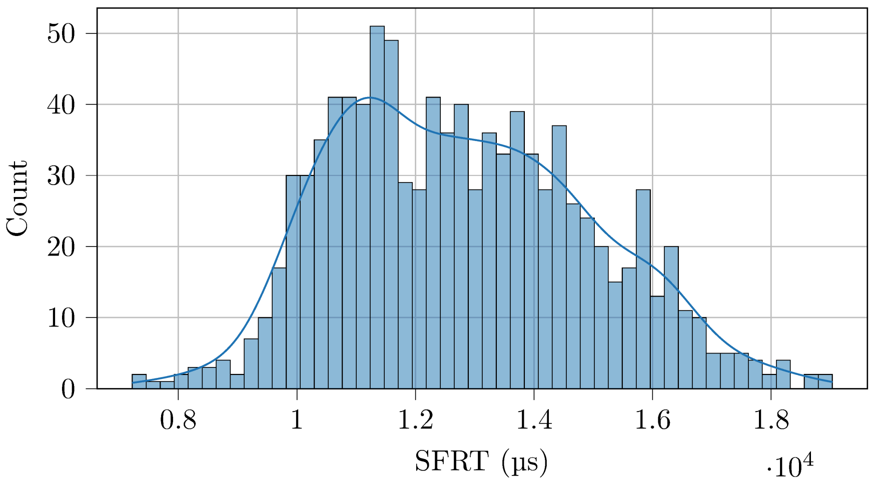


Figure 6.6: EPDF of the SFRT in the simulation with multiple mobile robots

a value could ensure that the robots are stopped on average in a very short time, guaranteeing the safety of personnel entering the automated area. However, as shown in Figure 6.5, there is a considerable amount of jitter in SFRT, which can, in some cases, lead to a considerable increase in SFRT. Despite this, in almost all simulations, the activation of the safety function occurs in approximately 20 ms at most. The worst case occurs when there is a packet loss. In the simulations carried out, a single case of packet loss occurred, resulting in an SFRT close to the theoretical limit, confirming what was seen in the simulations with fixed slaves. On the other hand, the maximum value obtained is also completely acceptable in the considered use case.

6.4.1 Discussion

The theoretical values obtained for the SFRT considering a watchdog time T_w set to 250 ms are sufficiently satisfactory for the considered application. Regardless of the type of polling scheduling strategy, it has been estimated that even in the worst case, i.e., with communication errors, the full stop of all robots in the operating area occurs theoretically within $330573.23\mu s$. As pointed out by Åkerberg et al. (2011) and in-depth in this chapter, the SFRT can be improved by using different polling strategies. Considering random polling strategy, it was possible to define a new estimate of the SFRT, namely $SFRT_P$ which value is definitely lower. This chapter shows through measurements on a calibrated simulator that $SFRT$ and $SFRT_P$ actually represent a theoretical upper bound reached only in the worst case. In fact, in most of the simulations performed, both with fixed and mobile nodes, the obtained SFRT is far lower than the estimated theoretical limit.

As seen in the theoretical discussion, the watchdog period has a significant impact on the value of the SFRT. The degree of safety on a system can be increased by decreasing the watchdog time, thus decreasing the SFRT value. By decreasing the watchdog time, the system will be more reactive to any communication problems and therefore to quickly force the entire system to a safe state. On the other hand, the drastic decrease can lead to usability problems of the system itself. Especially with wireless networks, which may be error prone and significantly more affected by interference with respect to wired connections, a watchdog period that is too low can lead to the watchdog timeout condition triggered much more often. The choice of the watchdog time and, therefore, of the SFRT must be a trade off between the required degree of safety and the usability to be guaranteed in the plant.

7 Assessment of Different OPC UA Implementations for Industrial IoT-Based Measurement Applications

In the last few years, the industrial world embraced the Industry 4.0 paradigm Lu (2017), which merges technologies with products, systems, and services having its intrinsic networked structures, to realize the Industrial Internet of Things (IIoT) (Sisinni, Saifullah, Han, Jennehag, and Gidlund (2018); Vitturi et al. (2019)). IIoT is a network of networks that connects industrial equipment, controllers, sensors and, actuators, i.e., the "Things", to provide diverse and advanced types of services in manufacturing systems, aiming at improving quality, productivity, efficiency, reliability, safety, and security.

Traditionally, the foundation of manufacturing and process industries has been in the deployment of specific distributed sensor systems for monitoring (and then controlling) the production process, thus leveraging the concept of Distributed Measurement Systems (DMSs) (Grimaldi and Marinov (2001)). With the increased pervasiveness of the IIoT paradigm, DMSs come even more into focus since the components of an IIoT system need an even further level of interaction to integrate instrumentation data, sensors, communication, and processing. Moreover, the need to improve production capacity and optimize the output process, eventually exploiting predictive maintenance and machine learning approaches (Witten, Frank, and Hall (2011)) requires an increased number of sensor devices and sensor swarms to be deployed.

This new IIoT–enabled DMS scenario is based on the support of efficient and reliable communication systems, ensuring widespread availability of data gathered from possibly heterogeneous measurement instruments and sensors (Tian (2001); Skrzypczak, Grimaldi, and Rak (2009)). Overall, the IIoT paradigm may represent the enabler for several enhanced measurement features: continuous and thorough measurements through low-power wireless connections, measurement collection over considerably broad geographic areas, and real-time analysis of measurement data collected from the field (Ooi and Shirmohammadi (2020)).

Unfortunately, in the highlighted IIoT scenario, it is expected that components and sensors devices come from different producers and use different formats to represent measurement data, and also it is very likely that they intrinsically operate over heterogeneous networks. Hence, the provision of ways to enable communication and interoperability among such devices is of paramount importance. A key solution toward this goal is the Open Platform Communication (OPC) Unified Architecture (UA) (Bruckner et al. (2019)). OPC-UA is a protocol defined by the IEC 62541 international standard conceived to implement Machine–to–Machine (M2M) communication over possibly different physical media while ensuring high level data protection against attacks and threats.

OPC-UA represents hence an appealing and advantageous opportunity for the arising IIoT measurement paradigm. Particularly, its object–oriented structure allows a complete contextualization of the information. For instance, an OPC-UA object could be used to store the value of a measurement, the features of the instrument/sensor, the measurement units, possible thresholds, and so on. Such important characteristics allow the new generation of measurement instruments to deal with multiple and heterogeneous data types.

At the same time, the complexity of the OPC-UA protocol may also reveal an obstacle to its introduction within measurement systems. Indeed, sensors and actuators, field equipment, and measurement instruments in the IIoT scenario are typically realized exploiting devices with limited hardware resources and low computational capabilities (and low costs). Consequently, the implementation of OPC-UA on such devices might be problematic and the performance might result compromised in terms of increased latency and power consumption, hence impairing the quality and accuracy of measurements.

7.1 Related Work and Contribution

The introduction of Industrial IoT technologies in the context of distributed measurement systems has started to be addressed some years ago. Paper by Ooi and Shirmohammadi (2020) deals with the potential of IIoT for the instrumentation and measurement fields. Notably, the authors provide an accurate assessment that addresses benefits and challenges, including also some useful commercial aspects. In Rizzi, Ferrari, Flammini, and Sisinni (2017), Lee and Ke, and Palisetty and Ray (2018), the authors address diverse Low Power Wide Area Networks (LPWAN) to enable IIoT-based measurement and monitoring applications over large distances. Both papers Bianchi, Boni, Fortunati, Giannetto, Careri, and De Munari (2020) and Liao and Lai (2020) describe the use of Wi-Fi, another important network for IIoT, to implement measurement systems that involve remote cloud data storage and analysis.

Moving to OPC-UA, in Cavalaglio Camargo Molano, Lahrache, Rubini, and Cocconcelli (2018) the authors describe a method to achieve synchronization among electrical drives connected via EtherCAT (a widespread real–time Ethernet network) using the OPC-UA protocol. Mainly, the paper deals with the important topic of obtaining a high accuracy synchronization over a geographically distributed system, which is of uttermost importance in distributed measurement applications. In Montavon, Peterek, and Schmitt (2019), the authors refer to metrology assisted assembly systems, and introduce the optical large–scale metrology instruments, such as laser trackers and indoor GPS. Then they propose an object–oriented model to formally describe such instruments and investigate the suitability of OPC-UA, as well as that of other protocols, to implement such a model. In Lee, Kim, and Lee (2017), OPC-UA is used to implement a smart sensor system to monitor the behavior of numerical control devices in the Industry 4.0 context. Specifically, OPC-UA objects are used to store sensor information such as measurement, threshold, range, product data, etc. Another interesting OPC-UA application is proposed in González, Calderón, Barragán, and Andújar (2017). Here the authors present a system based on OPC-UA to connect and integrate components typical of industrial automation as well as of distributed measurement systems. Examples of applications are provided that include smart microgrids, industrial laboratories, and energy systems in general. Paper by Ferrari, Flammini, Rinaldi, Sisinni, Maffei, and Malara (2018) considers an IIoT environment and focuses on transferring plant data to a cloud when OPC-UA–based gateways are used to gather data directly at the production level. Notably, the authors implemented a measurement system that allowed to determine the impact of Quality of Service parameters on communication delays.

The papers cited above represent interesting contributions. However, as far as OPC-

UA is concerned, they mostly describe meaningful applications that use such protocol without investigating its actual potential and capabilities. Nor do they consider protocol implementation aspects, which represent significant issues especially in the industrial scenario.

Moving from the above considerations, in this chapter, which substantially extends Morato et al. (2020), we propose a more structural assessment about the adoption of OPC-UA in the context of IIoT–based measurement systems. In this respect, we report and discuss the results of an experimental work on some popular implementations of the OPC-UA protocol stack. Particularly, we considered four OPC-UA implementations. Three of them are open source, namely Open62541, FreeOPC UA C++, and FreeOPC UA Python, whereas the fourth one is a proprietary product, namely Prosys OPC Java. The work aims to investigate the behavior of OPC-UA for the different implementations focusing on i) CPU usage, ii) communication times and iii) power consumption. In order to provide a meaningful and fair assessment, the protocol was implemented on a widespread commercially available Raspberry Pi Model 3B+ board that, thanks to its features, represents a manageable and effective testbed. The measurement setup has been designed to be of general usage and reproducible. Also, experiments have been mostly carried out using two widespread communication systems, namely Ethernet and Wi–Fi.

In detail, this chapter is organized as follows. Section 7.2 briefly describes the OPC-UA protocol and outlines the possible structure of distributed measurement systems that rely on OPC-UA. Section 7.3 introduces the experimental setup implemented for the measurements. Section 7.4 describes the tests carried out and discusses the obtained results. Finally, Section 7.5 concludes the chapter and outlines some future directions of research.

7.2 Introduction to OPC-UA

7.2.1 Origin and Motivation

The use of computers in industrial automation has been a growing trend since the early ’90s. One of the most significant difficulties in interfacing multiple components from different manufacturers is that each used a different communication system or proprietary protocols. The problem of creating a standardized system for accessing industrial process data was already known well before the advent of Industry 4.0. In 1995 a task force was created to define a Plug&Play technology that would allow Supervisory Control And Data Acquisition (SCADA) and Human Machine Interface (HMI) to access industrial process data in a standardized way. The result was the creation of Open Platform Communications (OPC) (henceforth referred to as OPC Classic), which allows the exchange of pure process variables. Initially, the operating principle was based on Microsoft's Component Object Model (COM) and Distributed Component Object Model (DCOM) technology designed for communication between software in networked computers. OPC Classic works as an Object Linking and Embedding (OLE) in which a Windows application is linked and communicates with industrial equipment. Industrial devices communicate through the same Windows machine using COM and communicate with other devices in the network using DCOM. While the use of standardized Microsoft technologies was a strength in the adoption of OPC Classic by providing a unique Application Programming Interface (API) for Windows systems, on the other hand, it was one of the main reasons why OPC Classic quickly gave way to the modern version by Open Platform Communications-Unified Architecture (OPC-UA). OPC Classic's biggest drawback was the restrictions on the Windows operating system. With the development of Ethernet based communication systems also in embedded devices and with the introduction to cloud–based computing, being bound to a single hardware/software platform was not a viable solution. Furthermore, the growing increase in size and complexity of industrial plants, more often composed of heterogeneous devices, required the exchange of more complex and structured information.

7.2.2 Open Platform Communication - Unified Architecture

Open Platform Communications-Unified Architecture (OPC-UA), formally released in its first version in 2006, is a vendor independent platform with an object–oriented architecture that integrates and extends the functionality of the original OPC Classic. The main features of OPC-UA are

- Backward compatibility: OPC-UA provides backward compatibility with OPC Classic functionality by replacing the COM–based system but without losing any of the featured or performance.
- Platform Independent: the entire OPC-UA architecture has been redesigned to be used in any hardware and software architecture. These include PCs, cloud–based services, and microcontrollers, whatever the operating system used.
- Security mechanism: The opening of OPC-UA towards internet and cloud–based services made it necessary to introduce encryption, authentication, and user control systems.

- Information Model: The information is structured in such a way as to give an easy semantic understanding of the structure of a complex system.

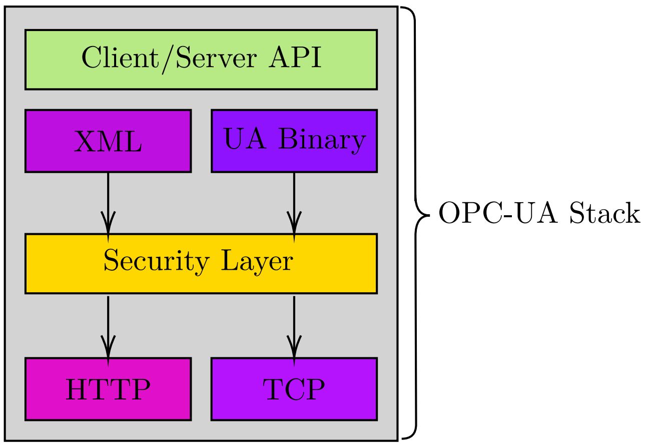


Figure 7.1: OPC-UA stack

The main elements that make OPC-UA highly interoperable and able to work in a large variety of systems are the transport mechanism and data modeling. In OPC-UA, the COM and DCOM transport protocols have been completely replaced by other transport protocols, namely TCP and Web Services, which represent the state of the art for reliable, secure, and platform independent communications. As shown in Figure 7.1, all messages coming from the API are encoded, encrypted by the security layer, and then sent with the transport mechanisms according to the encoding type that is being used. UA Binary represents the more optimized version of OPC-UA, where information is binary serialized and then sent as a TCP stream. This technique presents a low overhead, making it suitable for processing parameters between industrial embedded controllers. On the other hand, the traffic generated, which is usually encrypted and non-human-readable, can be blocked by a firewall, thus making communication via the internet difficult. To overcome this problem, another transport protocol called Web Services has been implemented, in which the information to be exchanged is encoded as an eXtensible Markup Language (XML) structure and then sent via SOAP/HTTP in the same way as Web browsers communicate to Web servers. Both encodings share the same security layer responsible for the encryption. It defines two security protocols, namely WS-SecureConversation and UA-SecureConversation, both based on a certificate-based connection establishment. WS-SecureConversation is mainly conceived to work with SOAP/HTTP and thus with the XML encoding. The major drawback is the relatively high overhead it introduces, which may represent a bottleneck in time-critical applications. In this regard, UA-SecureConversation represents a more suitable solution since the encryption based on TLS is more optimized.

Data modeling is necessary to define the basic rules with which OPC-UA exposes information. Indeed, OPC-UA is based on a client–server relationship, in which the information is structured following an object–oriented model, where objects are formally referred to as "nodes".

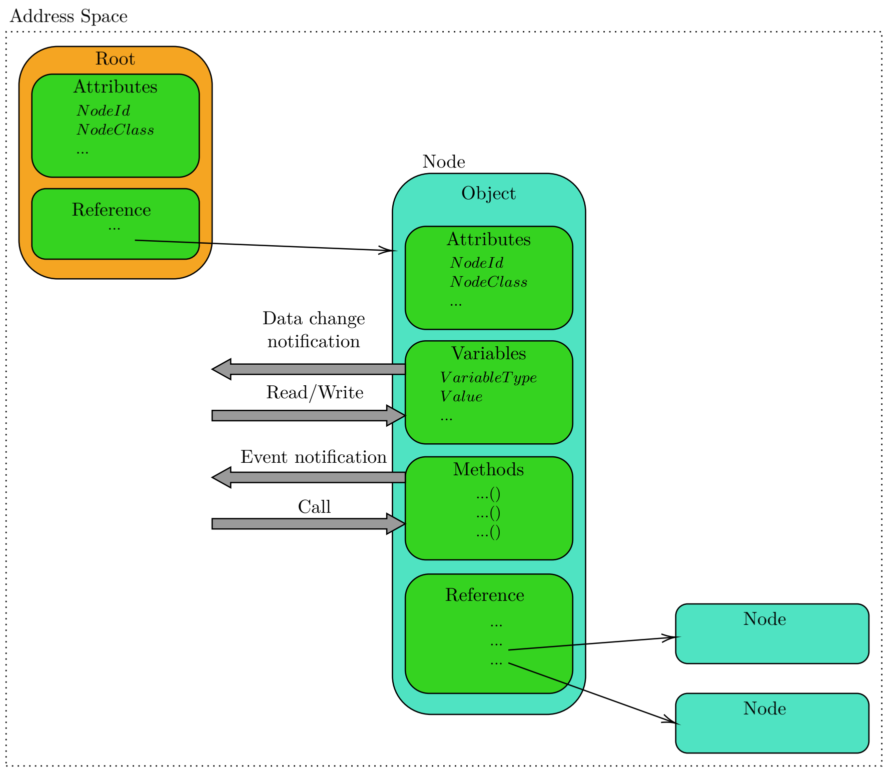


Figure 7.2: OPC-UA Information Model

As can be seen in Figure 7.2, conversely to OPC, where only pure data could be exchanged, the OPC-UA model defines the nodes' objects in terms of attributes, variables, methods, and references. A Node is the fundamental entity of OPC-UA and represents a basic object with only the attributes necessary to define any kind of information item (e.g., ID, name, etc.), reference to other objects, and methods that can be invoked. The set of nodes made available by an OPC-UA server is referred to as the address space (Damm, Leitner, and Mahnke (2009)). This structure, proper of the Object Oriented Programming (OOP), allows defining hierarchies and inheritance with which it is possible to arrange nodes in a way the final structure mimics a physical entity. In this way, the devices that want to access the device data can hierarchically navigate its structure, discovering what are the functions and data it exposes and their semantic description.

To access the information stored within the nodes, OPC-UA provides for a number of service sets. The *attribute* service set, in particular, contains the `read/write` services that a client may use to access the attributes of a node. Thus, for example, the attribute "value" of a node of type "variable" may be read (respectively, written) by a client by suitably invoking the `read` (respectively, `write`) service.

With the *subscription* service set, a client may define a "subscription" and assign to it some "monitored items". As summarized in Figure 7.3, any attribute of any node in the address space, such as variables or events, may be associated with a monitored item. For each monitored item in the subscription, it is defined the sampling interval T_S with which the server polls the data source for any changes. The acquired data or events are further processed by a filter that implements different rules to determine if and which data or events must be reported and which ones must be blocked. Data that passes the filter is placed in a queue, waiting to be sent. Based on a "publishing interval" T_P set by the client, the subscription periodically sends to the client the list of notifications that occurred during the publishing interval. In this way, for example, a client may be notified periodically with the list of changes of the attribute value of a node of variable type that occurred in the last publishing interval. The filter mechanism, therefore, has a rate limiter function. Since T_S may be faster than the rate T_P with which changes are notified, the filters make sure that only certain events or changes in variables are notified to the client, thus avoiding unnecessary bandwidth usage. On the other hand, the subscriptions are unique to each client, meaning that, for each client that wants to access data with this service set, there must be a unique instance of the subscription. As the number of clients grows, this leads to a noticeable increase in resource usage in terms of memory and CPU, which are often critical aspects in low-power embedded devices.

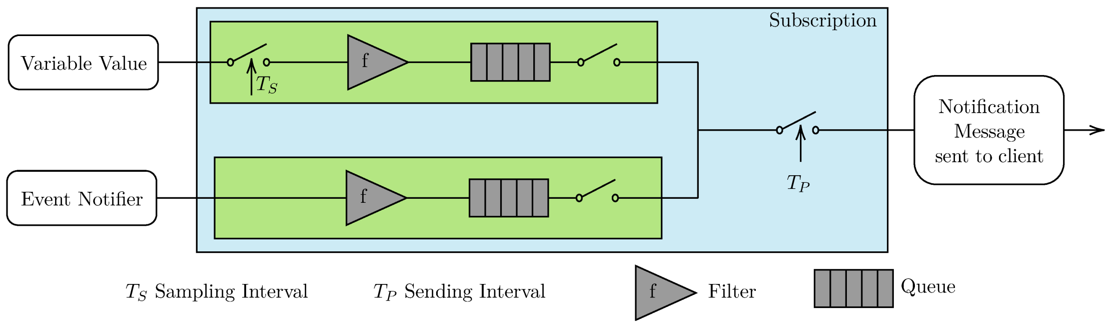


Figure 7.3: Monitored items in OPCUA. Inspired by Unified Automation

More recently, a different relationship to exchange information in the context of OPC-UA has been added to the standard, namely, the *PubSub* (Publisher – Subscriber) relationship. It is complementary to the client–server one. As shown in Figure 7.4, with this model, a publisher is responsible for creating "DataSets" that may contain the attributes of variable or event nodes (for example, the value of a variable). DataSets are grouped within “DataSetMessages“ that are sent to a “Message Oriented Middleware”, which, in turn, delivers them to subscribers. These ones, filter incoming messages, acquiring only those containing the topic to which they are subscribed. With such a model, it is possible to distribute data and events relevant to the publisher to a set of devices (subscribers) in a very efficient way, effectively making possible one-to-N, N-to-one, or even N-to-N communications. The transport and safety models have also been revised. OPC-UA PubSub uses UA Secure Multicast as its transport model, which is based on sending UDP Multicast frames with Private / Public Key based encryption.

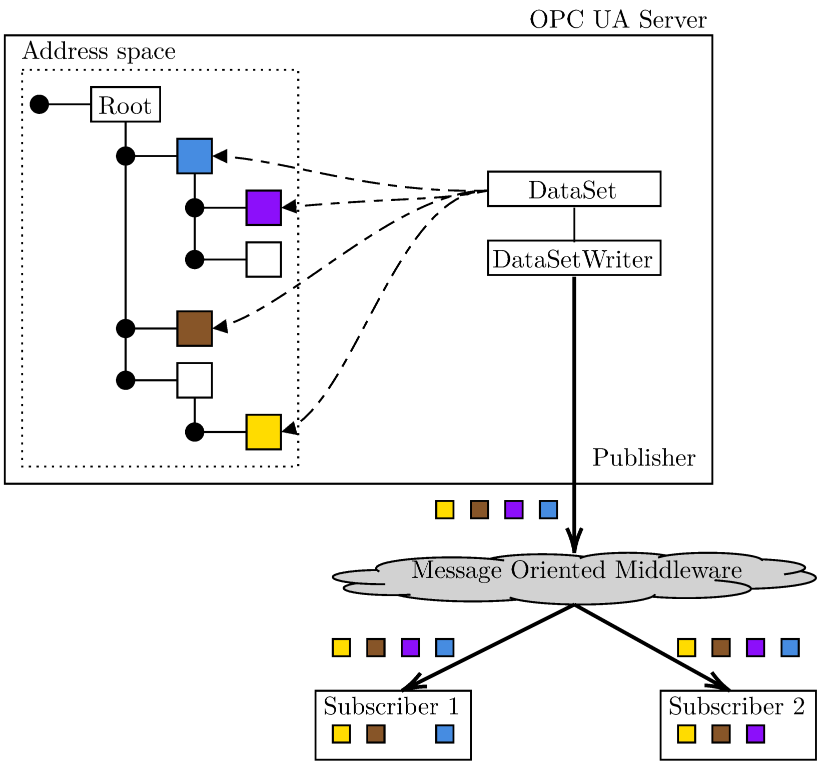


Figure 7.4: Publish/Subscribe in OPCUA.

An additional improvement is the integration of OPC-UA PubSub with Time-Sensitive Networking (TSN), enabling realtime communication from the field level to the cloud. In this regard, in classic fieldbuses or RTE networks is used a simplified version of the ISO/OSI concept to overcome complexity and performance penalty associated with the implementation of a full communication stack. In the context of OPC-UA, and more generally in all the IIoT scenario, this is not a viable solution, since the range of associated protocols on a converged IT and OT network will be significantly larger, requiring the ISO/OSI concept to be honored (Bruckner et al. (2019)). As can be seen in Figure 7.5, TSN enable seamlessly OPC-UA to operate in a real–time behavior while maintaining the full protocol stack. Furthermore, the combination of OPC-UA PubSub with TSN replicates, to some extent, the simplified ISO/OSI structure found in RTE networks, resulting in increased determinism and real–time capabilities.

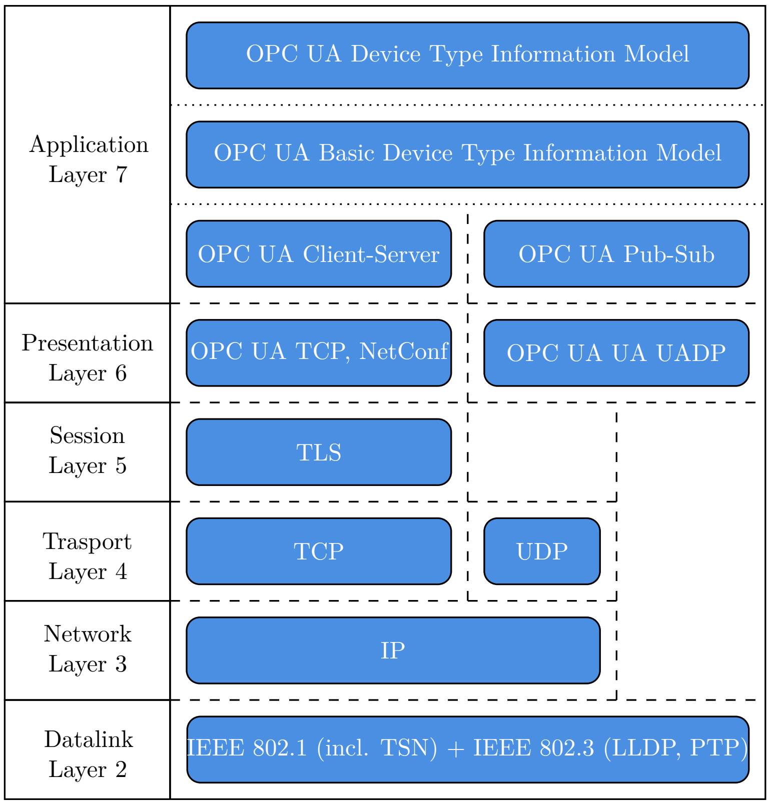


Figure 7.5: OPC UA TSN in the ISO/OSI layer reference model (Bruckner et al. (2019)).

7.3 OPCUA in distributed measurement applications

In the context of IIoT-based measurement applications, the OPC-UA protocol can be profitably exploited to allow seamless and secure interoperability among the heterogeneous sources of measurement data, and the heterogeneous communication networks. Indeed, measurements can be stored by nodes that belong to one or more servers so that they can be seamlessly accessed by distributed clients using, for example, the aforementioned service sets. An illustrative sketch representing the described scenario is reported in Figure 7.6. As can be seen, measurements stored in different devices, and structured within diverse OPC-UA servers, can be remotely accessed by an OPC-UA client, which implements techniques of real–time analysis and visualization.

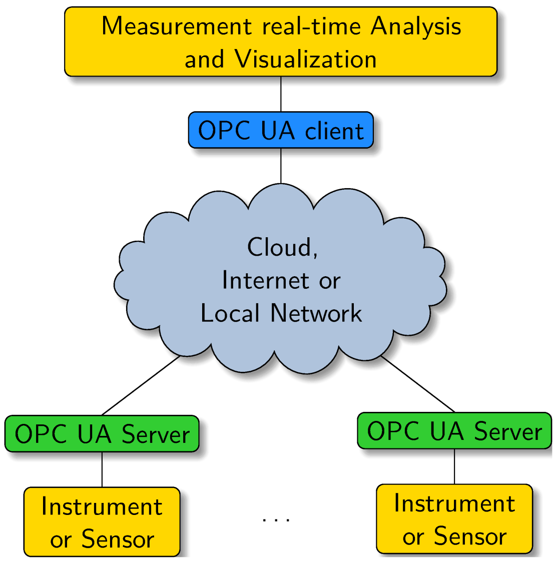


Figure 7.6: Example of use of OPC-UA in an IIoT-based Measurement System.

7.3.1 Experimental setup

The experimental setup has been designed to be as general as possible, with reproducibility in mind in order to be easily reused in different scenarios. The setup has been used to assess the behavior of some different OPC-UA protocol stacks, implemented on lightweight embedded systems, resembling those adopted by intelligent IoT sensors. Particularly, experiments have been carried out using Raspberry Pi Model 3B+ boards over two diverse communication systems, namely Ethernet and Wi-Fi, as schematically shown in Figure 7.7. Indeed, while both networks are meaningful in distributed measurement systems and IIoT scenarios, the former is much more targeted to high performance, ultra low-latency, and local measurement applications. In contrast, the latter represents its wireless counterpart, allowing for increased mobility and larger installations. More importantly, although Wi-Fi networks are able to provide very high transmission rate and good reliability, the performance figures they provide are clearly different with respect to those of Ethernet networks, which may be considered hence as a benchmark. An important aspect that will be analyzed in this work is the comparison between the two network supports in terms of latency to gather measurement data, which is a crucial parameter for accurate distributed measurements.

As can be seen, the same topology was used for the two networks, with OPC-UA client and server implemented on two diverse Raspberry Pi boards. A Netgear WGR 614 Wireless-G router was used to connect the two boards that was able to seamlessly implement the packet routing for both the Ethernet and Wi-Fi configurations.

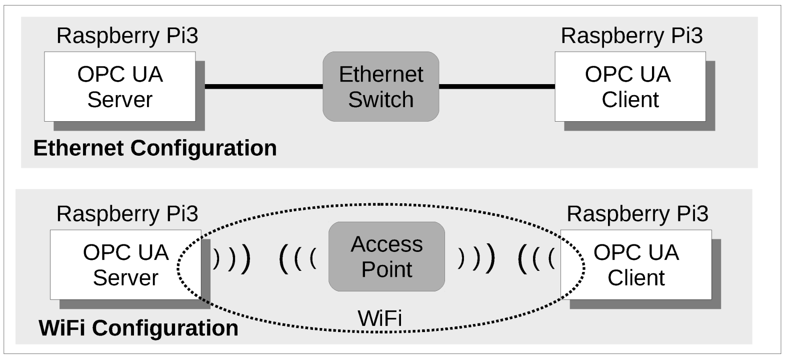


Figure 7.7: Experimental setup.

Two different operating system versions have been used for the boards. The first one is represented by the default Linux based Raspbian OS (Kernel version 4.14.79), whereas the second one is its real-time version (Kernel version 4.14.74–rt44). The latter one has been obtained from the default kernel version with the introduction of the `RT_PREEMPT` patch set, which enables a real–time behavior of the system allowing non critical parts of the kernel to be preempted in favor of the execution of userspace applications. Furthermore, we exploited a useful feature offered by the Linux operating system to isolate a group of CPU cores in which a process can be run. Specifically, the `isolcpus` boot parameter, in combination with the `taskset` command, allows to isolate one or more cores from the kernel scheduling and to reserve them for the execution of userspace applications without interference from the OS. Furthermore, in order to minimize the external factors that may affect the accuracy of the measurements, the CPU governor (i.e., a kernel–level component responsible for scaling the CPU frequency based on the workload) has been disabled during the experiments, and the CPU frequency has been set to its maximum operable value of 1.4 GHz.

Table 7.1: OPC UA Implementations.

OPC UA Implementation	Type	Programming Language	Commit Hash
Open62541	Open Src.	C99 / C++ 98	9f0c73d
FreeOPC UA C++	Open Src.	C++ 11	da2b76f
FreeOPC UA Python	Open Src.	Python	83fb9ea
Prosys Java v4.3.0-1075	Proprietary	Java	–

The implementations of OPC-UA considered in the experiments are reported in Table 7.1, along with the indication of the adopted programming language. For the open source implementations (available through the popular Github platform), we also provide the commit hash of the sources at the time the experiments were performed.

All the listed protocol stacks work natively on Raspberry Pi boards, and consequently, their setup procedures have not involved any further software adaptation. However, they are conceptually different. In particular, as can be seen, two out of four stacks are implemented using compiled languages (C/C++), whereas the other ones are implemented in, respectively, Python and Java, which are high level interpreted languages. As a consequence, the analysis of their behaviors reveals necessary to provide useful insights for the applications that use them. In this direction, since the outcomes of the experiments also depend on the adopted development environment, the most relevant technical details are summarized below:

- Python version 3.5.3;
- openJDK 1.8;
- glibc 2.23;
- gcc version 6.3.0;
- gcc optimization option: -O3 -s.

Finally, it has to be pointed out that other valuable OPC-UA implementations are available (either free of charge or commercial) that could have been considered in our work. For example, Eclipse Milo, ASNeG OPC-UA, OpenOpcUa, High Performance OPC-UA, to mention some. In this regard, an interesting overview of such solutions is provided in Haskamp, Meyer, Möllmann, Orth, and Colombo (2017). However, our analysis aims to investigate the behavior of OPC-UA in the context of IIoT based measurement systems, not to provide a comparison of different OPC-UA implementations. For this reason, we selected among the available solutions listed in Table II, paying attention to their diversity so that the proposed assessment can result adequately comprehensive.

7.4 Measurement Results and Analysis

The objective of the measurements is to assess the behavior of the diverse different OPC-UA implementations focusing on performance figures that are of interest for IIoT-based measurement instruments and applications. Specifically, we addressed the CPU usage, power consumption, and task execution times. The latter indicator is of particular significance in the application context of this chapter since it has reflects the overall latency with which measurement data can be collected at the client, and is hence a meaningful indicator of the intrinsic capability of the system to sustain real-time measurement analysis over networks (Ooi and Shirmohammadi (2020)).

7.4.1 CPU Usage

The first set of outcomes is resumed in Table 7.2, which shows the statistics about the CPU usage for the three considered implementations.

	CPU Usage		
	Mean	In Kernel Space	In User Space
Open62541	17.2%	60.89%	39.11%
FreeOPC UA C++	26.1%	50.63%	49.37%
FreeOPC UA Python	51.2%	–	–
Prosys Java	50.9%	–	–

Table 7.2: Statistics of the CPU Usage.

As can be seen, Open62541 is the most efficient from the average resources utilization point of view, followed by FreeOPC UA C++ and FreeOPC UA Python. The latter one highlighted a rather higher utilization compared to the other ones. However, this is not surprising since FreeOPC UA Python and Prosys Java are based on interpreted languages that are undoubtedly less efficient. It is interesting to note that, for the Open62541 implementation, the subdivision of the used resources of the CPU is slightly unbalanced towards the Kernel Space, while for FreeOPC UA C++, we have a subdivision almost at 50%. Unfortunately, values for FreeOPC UA Python and Prosys Java are not available because the tool `Perf`, with which the analysis was performed, does not support measurements of the stack of interpreted languages.

7.4.2 Read and Write Services

The experiments we carried out to test the OPC-UA read and write services were based on a purposely developed test OPC-UA task, with which the OPC-UA client reads an integer variable stored in the OPC-UA server. In the first session, we focused on open source implementations. In this task, the server implements two separate threads, as shown in Figure 7.8. Thread A simulates the acquisition of a new measurement (i.e., a physical quantity) every second by increasing an integer variable. Thread B is instead devised to manage the whole OPC-UA server. The measurement outcome, stored in an OPC-UA object, is saved in a memory area common to both threads so that the server can access it. In the test task, the OPC-UA client cyclically reads the variable's value stored on the server. This is accomplished by a read request issued by the client, to which the server answers in agreement with the OPC-UA protocol rules.

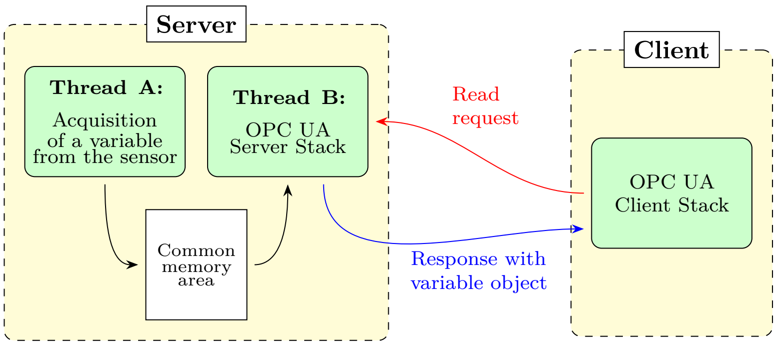


Figure 7.8: Test Task for OPC UA.

The outcomes relevant to the CPU usage for the read and write services are reported in Table 7.3. The table refers to context switches, CPU migrations, and the total number of CPU cycles to complete the execution of 100.000 consecutive OPC-UA test tasks. These outcomes are common indices exploited to determine the efficiency of a program, where high values indicate poor optimization and, therefore, long execution times. The results are in good agreement with those shown in subsection 7.4.1. Also in this case, both the compiled implementations have comparable values, whereas both the Python and Java based ones show much higher values regarding, in particular, the number of CPU cycles.

	context switches	CPU migrations	CPU cycles
Open62541	$100 \cdot 10^3$	1	$7.7 \cdot 10^9$
FreeOPC UA C++	$200 \cdot 10^3$	0	$13 \cdot 10^9$
FreeOPC UA Python	$283 \cdot 10^3$	29	$533 \cdot 10^9$
Prosys Java	–	–	$174 \cdot 10^9$

Table 7.3: CPU usage for the OPC UA test task.

To assess the performance of the read and write services, we measured the *task execution time*, T_s, defined as the time necessary to complete one instance of the OPC-UA test task described in Figure 7.8. Specifically, T_s represents the time that elapses between the read request of the client, T_{req}, and the time at which it actually receives the OPC-UA object containing the variable, T_{res}.

$$T_s = T_{res} - T_{req} \tag{7.1}$$

In the experiments, T_s has been measured by direct access to the content of the *Cycle Counter Register* (CCR), an internal CPU register implemented within ARM processors, which is a counter of the processor clock cycles. This design choice is significant to improve the accuracy relevant to the measurement of the task execution time because accessing the CCR register requires only one CPU cycle as specified in the Arm Architecture Reference Manual, hence introducing a negligible impact on the evaluation of the time T_s. The OPC-UA test task was run continuously, meaning that a new instance of the task was started immediately after the conclusion of the former one. In the Ethernet configuration, the selected transmission rate was 100 Mbit/s whereas, for the Wi-Fi one, we chose the IEEE 802.11g mode, with transmission rate dynamically selected by the multi–rate support feature provided by such protocol[1]. For each experimental session, $N = 100.000$ measurements of the execution time have been collected and analyzed. Furthermore, all the components of the experimental setup were located sufficiently close to each other to ensure, particularly for the Wi-Fi configuration, a high success probability in packet delivery. This has been subsequently confirmed by the traffic analysis we carried out, that showed a very low number of packet retransmissions and losses, with an average of about 3.6%.

The statistics of the execution time for the OPC-UA test task are reported in Table 7.4 for the case of non–isolated CPU and, respectively, in Table 7.5 for the isolated one.

		Execution Time T_s [µs]			
		Generic OS		RT OS	
		Mean	Std	Mean	Std
Ethernet					
	Open62541	312.83	12.56	382.48	24.44
	FreeOPC UA C++	377.30	4.54	467.59	15.01
	FreeOPC UA Python	736.79	7.44	778.43	20.63
Wi-Fi					
	Open62541	2036.12	501.60	3765.62	924.52
	FreeOPC UA C++	2063.38	488.10	3869.62	907.36
	FreeOPC UA Python	9274.24	750.59	12824.94	3901.25

Table 7.4: Statistics of the Execution Time for the OPC UA Test Task – Non–Isolated CPU.

At first glance, the beneficial effect of introducing CPU isolation appears clear. Indeed, as shown in Table 7.5, all mean values decrease when such a feature is used. The behavior of standard deviations is similar, with an exception relevant to the Open62541 implementation. In this case, a slight increase (from 12.56 to 12.76 µs) is observed

[1]Please note that the multi–rate support feature could not be disabled on Raspberry Pi boards.

		Execution Time T_s [µs]			
		Generic OS		RT OS	
		Mean	Std	Mean	Std
Ethernet					
	Open62541	306.67	12.76	368.78	10.13
	FreeOPC UA C++	374.74	3.32	457.60	11.32
	FreeOPC UA Python	711.27	6.52	735.84	9.69
Wi-Fi					
	Open62541	1950.29	460.55	3431.68	886.28
	FreeOPC UA C++	2017.71	457.51	3656.67	902.52
	FreeOPC UA Python	9031.62	713.10	12463.11	3807.16

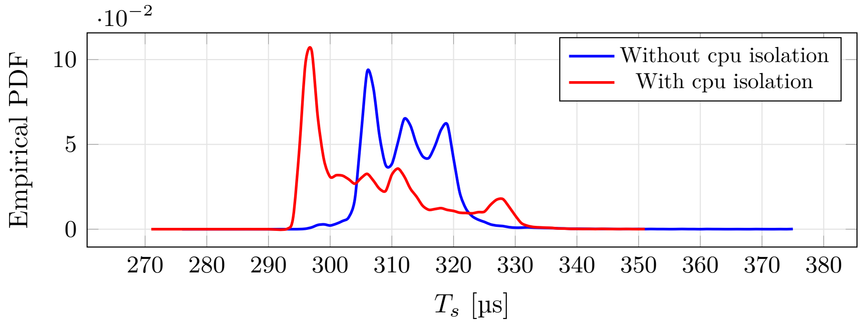
Table 7.5: Statistics of the Execution Time for the OPC UA Test Task – Isolated CPU.

when switching from the non–isolated case to the isolated one. We believe this aspect may be explained by the low average execution time of the Open62541 implementation. Indeed, we have checked that only the 30% of the CPU time was necessary to execute the Open62541 stack. Thus, both the task execution time and its variability are mostly determined by the execution times of unbounded kernel threads (e.g., those concerned with the TCP/IP protocol suite, the network drivers, etc.), as well as by the network transmission times, that do not benefit from the CPU isolation.

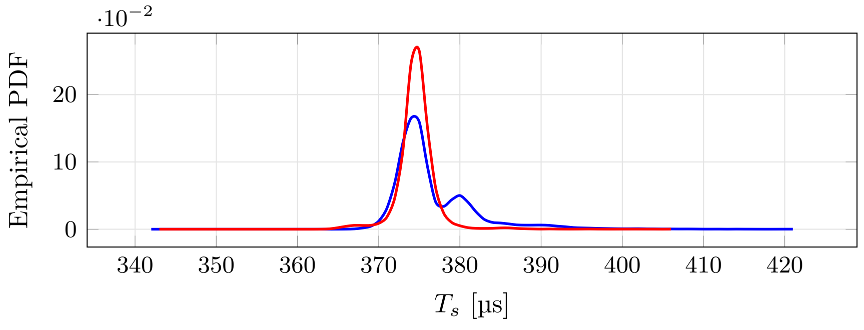
The results shown in Table 7.5 are confirmed by the probability density functions obtained experimentally Empirical Probability Density Function (EPDF)), reported for the Ethernet configuration, respectively, in Figure 7.9 (generic operating system) and in Figure 7.10 (RT operating system). As can be seen, the CPU isolation improves the EPDF shapes in most cases.

The tests on the Wi-Fi configuration revealed similar trends, as shown in Table 7.4 and Table 7.5 and both Figure 7.11 and Figure 7.12. In particular, the benefits brought by the CPU isolation are evident. As can be seen, both mean and standard deviation are greater with respect to the Ethernet case. This is due to the longer packet transmission times as well as to the possible retransmissions (interleaved by random backoff times) that could be necessary to successfully deliver a packet. Also, as Seno, Tramarin, and Vitturi (2012) discussed, the task execution time is further negatively influenced by the Access Point, which may introduce additional, non negligible, delays and randomness.

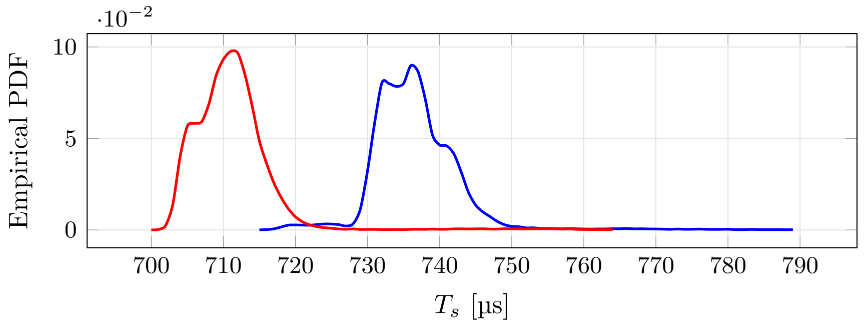
The introduction of the RT operating system, in general, worsened the behavior of the OPC-UA test task execution time, as can be evinced from the statistics and the EPDF reported above. Actually, the mean values are higher for all the OPC-UA implementations with respect to the correspondent cases in which the generic OS is used. The same happens for the standard deviation, with the unique exception of the



(a) Open62541



(b) FreeOPC UA C++

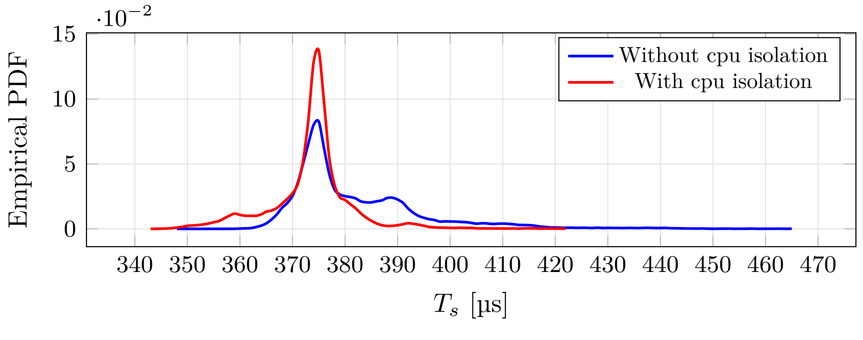


(c) FreeOPC UA Python

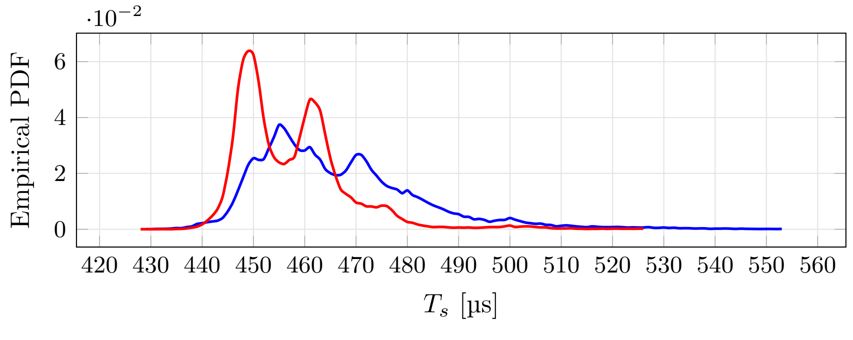
Figure 7.9: Generic OS: EPDF of the execution time of the OPC UA test task for the Ethernet configuration. Blue line: configuration without CPU isolation. Red line: configuration with CPU isolation enabled, where both server and client are forced to run on the isolated CPU.

Open62541 implementation (in this case, the value decreases from 12.76 to 10.13 µs when the RT OS is used). However, as already pointed out, these values, for the Open62541 implementation, are mostly influenced by the times necessary to execute unbounded kernel threads.

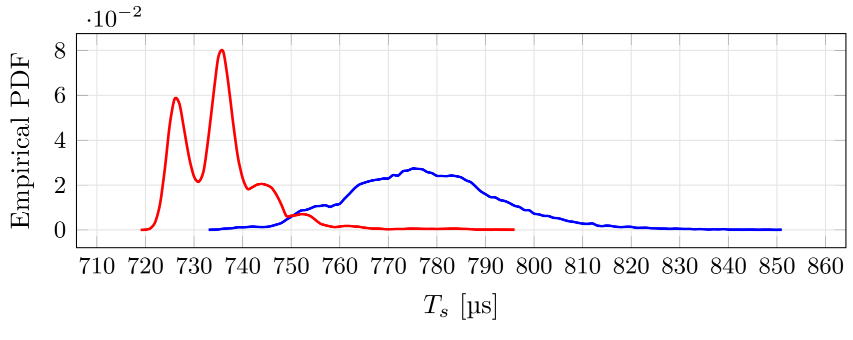
Although the worsening observed when the RT operating system is used may seem surprising, it may be explained by making some considerations about the introduction of



(a) Open62541



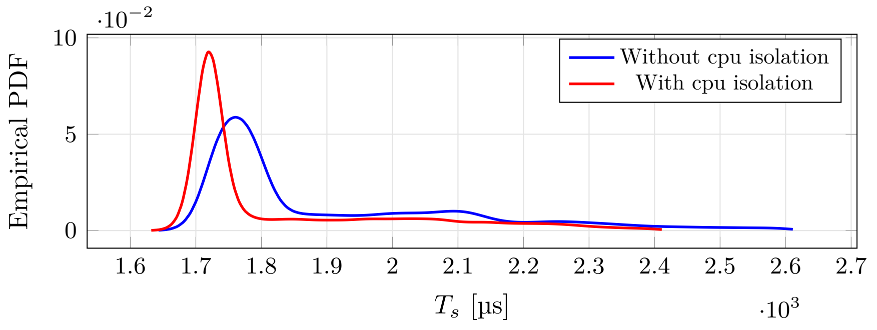
(b) FreeOPC UA C++



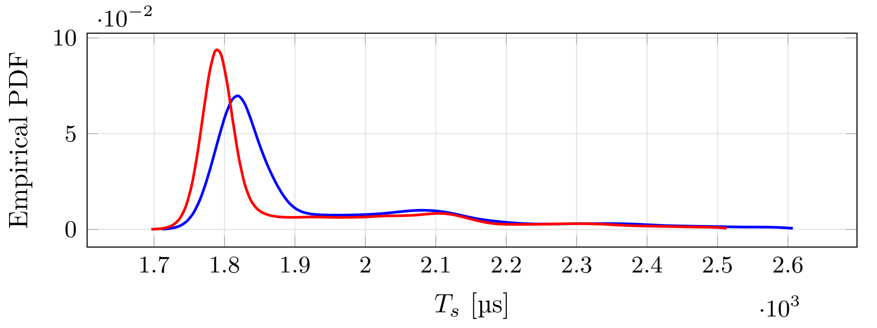
(c) FreeOPC UA Python

Figure 7.10: RT OS: EPDF of the execution time of the OPC UA test task for the Ethernet configuration. Blue line: configuration without CPU isolation. Red line: configuration with CPU isolation enabled, and where both server and client are forced to run on the isolated CPU.

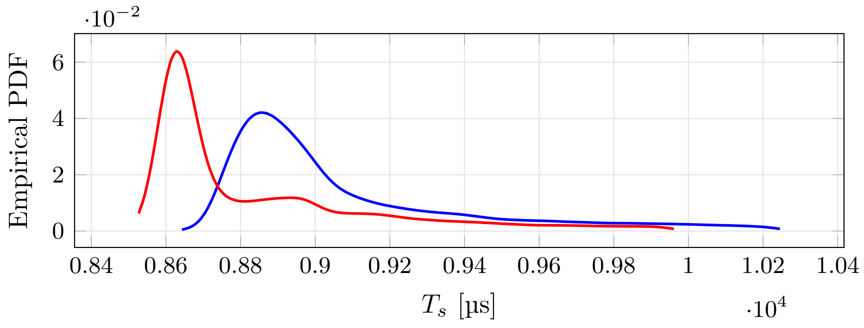
the Linux real–time extension. As shown in Table 7.2, all the considered implementations of the OPC-UA protocol stack make extensive use of the kernel functions, especially those concerning network connectivity. Nonetheless, the real-time patch makes some parts of the kernel preemptable, thus leaving more space for executing instructions in the userspace. Thus, the stack execution could be interrupted more frequently, resulting in longer execution times and greater jitter. Furthermore, as reported in LeMaRiva|Tech,



(a) Open62541



(b) FreeOPC UA C++

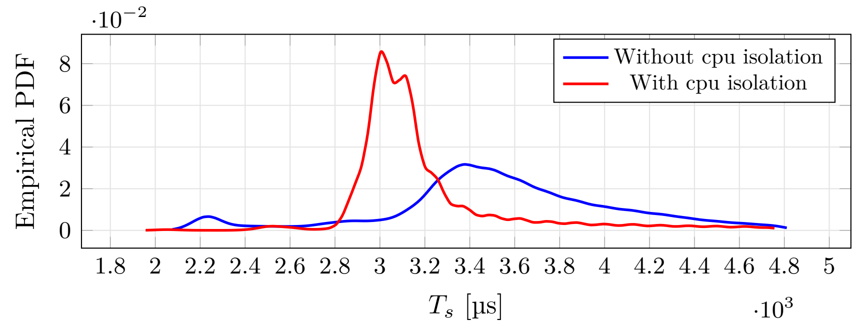


(c) FreeOPC UA Python

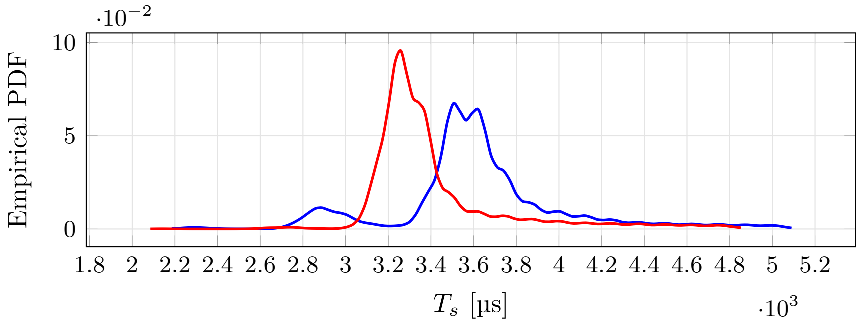
Figure 7.11: Generic OS: EPDF of the execution time of the OPC UA test task for the Wi-Fi configuration. Blue line: configuration without CPU isolation. Red line: configuration with CPU isolation enabled, where both server and client are forced to run on the isolated CPU.

the application of the Linux real–time extension has negative effects on the throughput of the communication interface. This is confirmed by the traffic analysis that showed, in the worst case, an increment of 2.8 times of packet retransmissions.

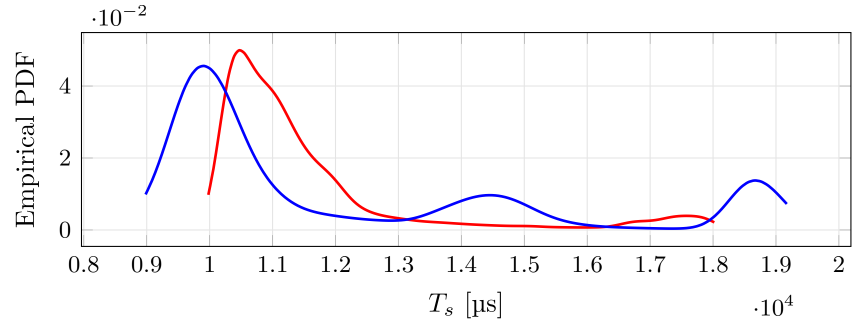
Focusing on the stack implementations, the obtained results show that both the compiled versions, Open62541 and FreeOPC UA C++, are characterized by comparable average values of the OPC-UA test task execution time. Conversely, the average T_s



(a) Open62541



(b) FreeOPC UA C++



(c) FreeOPC UA Python

Figure 7.12: RT OS: EPDF of the execution time of the OPC UA test task for the Wi-Fi configuration. Blue line: configuration without CPU isolation. Red line: configuration with CPU isolation enabled, and where both server and client are forced to run on the isolated CPU.

is much higher (about doubled) for the FreeOPC UA Python implementation, as was expected since Python is an interpreted language. This aspect is exacerbated for the Wi-Fi configuration. As far as the standard deviation is concerned, with the Ethernet configuration, Open62541 presents higher values than both FreeOPC UA C++ and FreeOPC UA Python, especially as percentage of the mean, reflecting in a considerable jitter of the execution time. This feature is evident for the generic operating system, whereas it appears more vague for the RT operating system, likely due to the additional randomness introduced by this latter one. A similar consideration can be made for the Wi-Fi configuration. In this case, as can be seen, both the mean and standard deviation are increased with respect to Ethernet. However, the standard deviation values become comparable, especially for Open62541 and FreeOPC UA C++, likely as an effect of the randomness in accessing the physical medium introduced by Wi-Fi.

The final experiments of the `read` and `write` services were concerned with the Prosys Java implementation. Due to space limitations, we considered only the (most meaningful) case of generic OS over Ethernet. The EPDF of the test task execution time is shown in Figure 7.13, whereas the statistics are reported in Table 7.6. As can be seen, both mean and standard deviation are considerably higher with respect to the open source implementations, confirming the trend already observed when an interpreted language (Java in this case) is used. Also, the EPDF shown in Figure 7.13 evidence the high variability of T_s, which is only partially mitigated by the CPU isolation.

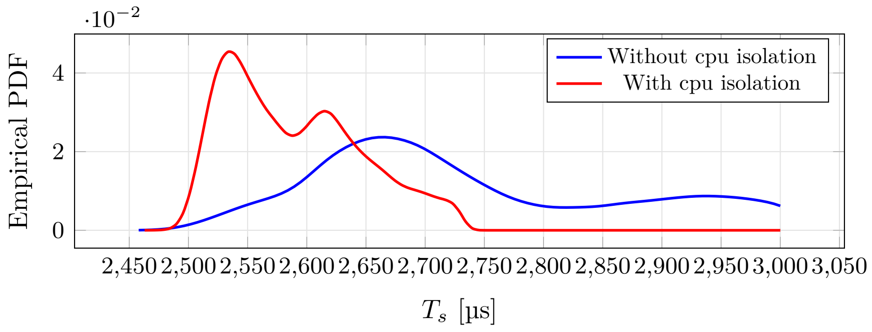


Figure 7.13: EPDF of the execution time of the OPC-UA test task for the Prosys Java implementation on Ethernet with Generic OS.

Execution Time T_s [µs]			
Without CPU Isolation		CPU Isolation	
Mean	Std	Mean	Std
2793.19	348.50	2648.31	280.13

Table 7.6: Statistics of the Execution Time for the OPC-UA Test Task for the Prosys Java implementation on Ethernet with Generic OS.

7.4.3 Subscription Services

In this session of experiments, we addressed the OPC-UA subscription services. Similar to the last test of the previous subsection, we considered the case of generic OS over Ethernet. Moreover, we focused on only one open source implementation, namely Open62541, and on the Prosys Java proprietary solution. Notably, Open62541 revealed the most effective open source implementation among those addressed in the previous subsection. Also, as discussed in Palm, Gruener, Pfrommer, Graube, and Urbas (2015) and Pfrommer (2016), Open62541 is well supported and suitable for applications in the IIoT scenario.

Subscription services are completely asynchronous and let the server notify changes in its nodes. Referring to Figure 7.8, we associated a monitoring item to the sensor value simulated by Thread A. In this way, the server checks the data source with a period defined by the "Sampling Interval" and, when the publishing interval elapses, it sends a "Publish Response" message containing the notification of data change. Then the client acquires the data and sends a "Publish Request" message that acknowledges the received data.

To assess the performance of the subscription service for the Open62541 implementation, we evaluated the *delivery time*, defined as the time employed by the client to acquire data published by the server. The measurements have been carried out as follows. When the publish response message is ready to be transmitted, the server sets one of the GPIO pins of the Raspberry Pi. Similarly, when the client receives the message at the application level, it sets one of its GPIO pins. The interval elapsed between the generation of the two consecutive signal edges represents the delivery time. This type of measurement also has the advantage of not requiring the systems to be time synchronized. In this experiment, 6000 measurements have been collected and analyzed.

On the server, both the publishing interval and sampling interval have been set to 100 ms, which is the minimum selectable value on the default implementation of Open62541, whereas the update interval of Thread A has been set to 30 ms. It is important to note that these values are not imposed by the OPC-UA standard, but rather by the implementations to limit computational complexity. The signal edges of the GPIO have been acquired by a logic analyzer with a sample rate of 24 MHz.

The EPDF of the delivery time is shown in Figure 7.14, whereas its statistics are provided in Table 7.7. Although an effective comparison with the results of the read and write services reported in Subsection 7.4.2 can not be done (the adopted measurement techniques had to be necessarily different), it may be observed that the delivery time has, on average, lower values than the task execution time of the read service. This can be explained considering that, with the subscription services, a single message transmission

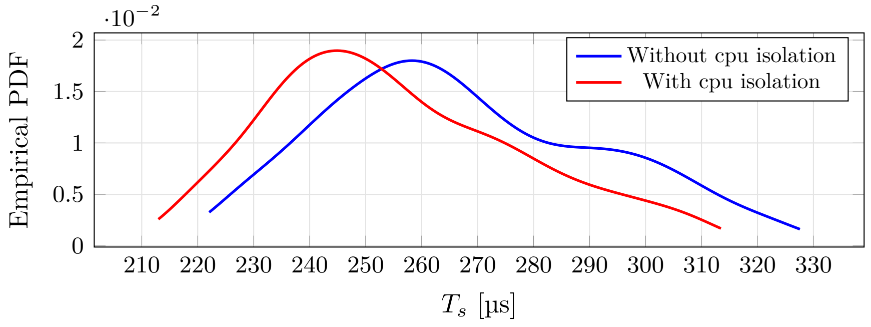


Figure 7.14: EPDF of the delivery time for the subscription service – Open62541 implementation over Ethernet with Generic OS.

Delivery Time [µs]			
Without CPU Isolation		CPU Isolation	
Mean	Std	Mean	Std
267.77	24.55	255.68	22.59

Table 7.7: Statistics of the delivery time for the subscription service Open62541 implementation over Ethernet with Generic OS.

by the server is sufficient to make the measured data available to the client. Conversely, the jitter increases with respect to the read service for the Open62541 implementation. For the same considerations made in Subsection 7.4.2, we believe this is most likely due to the execution of unbounded kernel threads that may heavily impact the behavior of the delivery time. As a final observation, the CPU isolation is also beneficial in this case.

Moving to the Prosys Java implementation, an analogous procedure to evaluate the delivery time could not be used since, for such a proprietary implementation, it has not been possible to modify the stack core to set the GPIO pins. Thus, we resorted to analyze the timestamps of the messages exchanged over the network, acquired with Wireshark. The delivery time has been hence determined as the time interval between the generation of the publish response message and the arrival of the acknowledgment generated by the client.

The EPDF of the delivery time is shown in Figure 7.15, whereas the statistics are reported in Table 7.8.

As can be seen, also in this case, the subscription service appears more efficient than the read one (although, as already pointed out for the Open62541 implementation, an effective comparison can not be made). Indeed, the mean time necessary to acquire a measurement by the client is definitely lower than that shown in Table 7.6 for the read service. Also, the beneficial effect of the CPU isolation is evident.

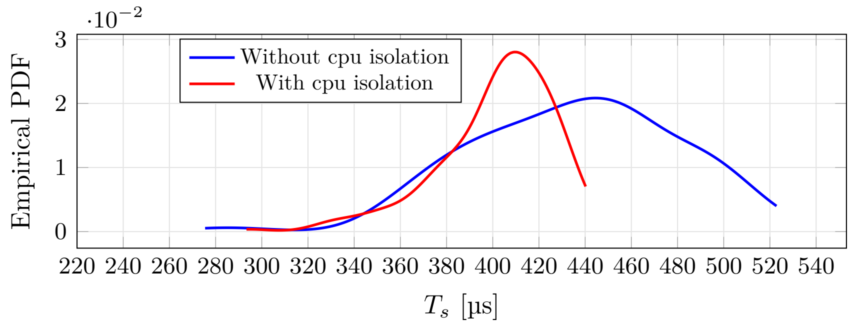


Figure 7.15: EPDF of the delivery time for the subscription service – Prosys Java implementation over Ethernet with Generic OS..

Execution Time T_s [µs]			
Without CPU Isolation		CPU Isolation	
Mean	Std	Mean	Std
434.33	47.57	401.53	24.84

Table 7.8: Statistics of the delivery time for the subscription service Prosys Java implementation over Ethernet with Generic OS.

7.4.4 PubSub Communication Profile

In a final experimental session, we addressed the OPC-UA PubSub communication model. Currently, PubSub is only supported by Open62541 and, although it provides all the main methods, it is an experimental version still under heavy development. Similarly to the subscription services carried out for Open62541, the server has been configured to publish a message every 100 ms. These messages were sent to a message oriented middleware via UDP multicast transmissions using the UADP encoding rather than using the full stack implementation which, as previously discussed and shown in Figure 7.5, should theoretically decrease communication time and improve real–time capabilities. As for Open62541, we measured the delivery time as the difference between the generation of the two consecutive signal edges on the GPIO pins. The EPDF of the delivery time and its statistics are reported respectively in Figure 7.16 and Table 7.9. Contrary to expectations, with the use of PubSub, there is a remarkable worsening of performance with respect to the subscription services. Indeed both mean and jitter values increase considerably. This is an unexpected result since the PubSub communication model has been conceived to minimize protocol overhead and hence to reduce the transmission times between publishers and subscribers. Thus, there are no logical explanations for these outcomes. We suppose the problem is because the PubSub implementation of Open62541 is still under development.

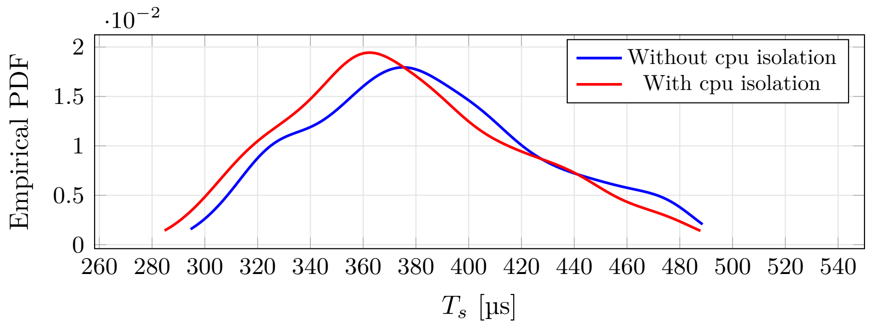


Figure 7.16: EPDF of the pubsub transmission time for the Open62541 implementation on Ethernet with Generic OS.

Delivery Time T_s [µs]			
Without CPU Isolation		CPU Isolation	
Mean	Std	Mean	Std
384.77	43.93	376.28	43.69

Table 7.9: Statistics of the delivery time for the pubsub communication profile over Ethernet with Generic OS.

7.4.5 Power Consumption

One of the main issues concerned with, possibly mobile, battery powered devices is the autonomy. Indeed, such devices have to ensure a good level of performance for a given amount of time. To meet these requirements, modern processors are capable of DVFS to minimize their power consumption and, consequently, extend the battery lifetime (Howard, Dighe, Vangal, Ruhl, Borkar, Jain, Erraguntla, Konow, Riepen, Gries, Droege, Lund-Larsen, Steibl, Borkar, De, and Van Der Wijngaart (2011)). In the Raspberry PI boards used in the experimental setup, the DVFS functionality is driven by a default kernel governor, called `ondemand`, that dynamically adjusts the CPU frequency according to the workload variation. Specifically, if the workload exceeds a predefined threshold for a certain amount of time, then the governor increases the CPU operating frequency to its maximum value. Conversely, if the workload is below the threshold, the operating frequency is switched to the lowest feasible one (Karpowicz (2016)). This approach clearly, represents an optimal trade-off between performance and power consumption in a generic processing system.

We analyzed the DVFS impact on both power consumption and performance for the experimental setup considered so far. The first tests have been performed using the Open62541 stack, on the Wi-Fi configuration, with the generic operating system, and without CPU isolation using read/write services. In detail, we measured the current consumption on the client side, and the time necessary to complete the experimental session described in Subsection 7.4.2, that comprised the execution of $N = 100.000$ OPC-UA test tasks.

The circuit implemented for current measurement is described in Figure 7.17. The Raspberry Pi was powered by a stabilized power supply, providing a 5 V continuous voltage. The adsorbed current was measured using a Hall effect sensor whose sensitivity is 185 mV/A. Current measurements have been acquired using an external digital acquisition system equipped with a 12 bit ADC with an input range of 0 V to 3.3 V at a sampling rate of 1 kHz. Each time the OPC-UA test task is started, the Raspberry Pi raises a signal triggering a new acquisition of the current level, which is also timestamped for further analysis.

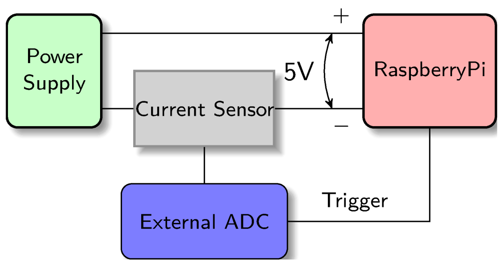


Figure 7.17: Setup adopted for current measurements.

	Current [A]			**Session completion time** [ms]
	Mean	Std	Idle	
Governor Enabled	0.4175	0.0587	0.3928	235924
Governor Disabled	0.4767	0.0760	0.4462	215259

Table 7.10: Statistics of the current consumption with CPU governor disabled and enabled.

The results of this new set of experiments are summarized in Table 7.10, which reports the statistics of the current consumption for the two cases in which the governor was, respectively, enabled and disabled. The table also shows the current consumption in idle state (i.e., while the experimental session was not carried out) for comparison.

As can be seen, enabling the governor leads to a slight decrease in current consumption for both average and standard deviation values. However, the time necessary to complete the experimental session increases, by almost 10%, as a result of the continuous CPU frequency adjustments caused by the workload variations. Thus, at first glance, it might not be worth maintaining the governor enabled since the benefits achieved in terms of power savings may result nullified by the performance degradation. However, a decision in this direction has to consider other aspects, such as the specific devices adopted and the performance requirements.

A further observation can be made with respect to the current consumption in idle state. Table 7.10 clearly shows only a limited increase of the current consumption when moving from this state to that in which experiments were executed, regardless of the governor status (enabled or disabled). This may be explained considering that Open62541 uses very low CPU resources that, evidently, are not sufficient to imply a remarkable variation in current consumption, as confirmed by the results presented in Table 7.2.

Finally, we carried out additional tests for the subscription service set and PubSub communication profile. In both cases, we measured the power consumption for the Open62541 implementation, over Ethernet, with the governor disabled. The analysis, actually, has not revealed any significant change with respect to the former measurements.

7.5 Conclusion and Future works

In this chapter, we considered the case of IIoT measurement applications and proposed the adoption of the OPC-UA protocol to enable seamless interoperability when heterogeneous sources of measurement data coexist, sharing information over the network. We focused on four widespread implementations of the protocol to analyze their impact on a networked measurement system, mainly in terms of latency and power consumption. A reproducible and effective measurement setup has been designed that allowed to carry out a thorough assessment, thus providing a complete characterization of OPC-UA for network-based measurement instrumentation systems.

Meaningful results have been obtained, also allowing to provide some interesting implementation guidelines. For instance, it has been verified that using a real-time operating system does not bring specific advantages whereas, in general, the best performances are achieved with a generic operating system exploiting the CPU isolation for the measurement application.

Furthermore, in compliance with modern IIoT-based applications, we considered the case of battery powered wireless measurement systems, thus providing some valuable insights about the expected power consumption in some selected and relevant cases.

Indeed, the measurement campaign highlighted that the DVFS feature should be enabled, allowing for lower power consumption without compromising the performance.

The current work opens up to future analysis. For instance, power consumption also depends on intrinsic parameters of the accumulator and environmental conditions, hence requiring a more extensive experimental campaign focused on mobile battery powered measurement instruments. In addition, the proposed experimental setup for latency analysis seems to be overkill for small integrated smart sensors. Hence, we plan to test the framework on low power embedded devices, like widespread microcontrollers with no operating systems.

8 Conclusions

The IIoT ecosystems are rapidly expanding towards devices and applications requiring strict timeliness, a relatively high safety, and data contextualization. Consider, for example, autonomous collaborative robots whose safe operation is determined by a correct exchange of process data, often wirelessly, and the coordination between distributed sensors and controllers. These features require the development and deployment of high-performance communication networks capable of interconnecting many distributed devices. Furthermore, the need for increasingly strict safety requirements may easily represent a bottleneck for the large scale adoption of Industry 4.0 concepts. Thus, there is a growing necessity in providing and assuring a real-time exchange of information to guarantee safe operations. Moved by these considerations, our contribution mainly focused on studying the challenges in IIoT enabled safety communication systems and distributed measurements systems.

In *Chapter* 2, we illustrated how the panorama of safety communication protocols in the industrial sector is decidedly broad. Unfortunately, most industrial RTE networks include their own safety extension, none of which are originally intended to operate over wireless networks. On the other hand, the design characteristics of most of these protocols open up the concrete possibility of being able to use them on means of communication other than those for which they were designed. Indeed, the functional safety protocols defined by IEC 61784-3 have been designed assuming the underlying communication system behaves like a *black channel*. With such an approach, the safety nodes are not aware of the channel features, nor of the industrial network they rely on, except for the service primitives necessary to data transmission.

From these assumptions, in *Chapter* 3, we investigated the possibility of using Fail Safe over EtherCAT (FSoE) on IEEE802.11. The safety protocol stack was implemented and tested on two experimental setups, the first with embedded devices without an operating system, while the second was on devices equipped with an operating system and higher computational capacity. In both cases, the results of the experiments confirm the potential effectiveness of the black channel approach, as they demonstrate the possibility of implementing the FSoE protocol on a network for which it was not explicitly designed, without any modification to the protocol itself. This achievement can be likely extended to the other protocols that adopt the same approach, e.g., those defined by IEC 61784-3. Although the polling time proved adequate for soft real–time applications, the number of lost packets was too high, concluding that the SIL3 can not be reached. The use of a transport protocol such as UDP that does not include any flow control and quality of service management is certainly one of the causes of the high packet loss.

In *Chapter* 4, we investigated the use of other transport protocols for the encapsulation of safety PDUs. While TCP seems like the obvious choice from this point of view, the lack of broadcast addressing could make it difficult to use in some applications. We have therefore proposed a protocol agnostic caching layer that implements the retransmission and management of duplicate frames. In our implementation, the layer was interposed between UDP and FSoE. The effectiveness of these protocols was assessed by measuring the number of lost packets in a simulated and real transmission channel, as well as monitoring the impact on the polling time. Both TCP and caching layer implementations have outperformed UDP, drastically lowering packet loss. The impact of TCP on the polling time was more evident, while the introduction of the caching layer has practically irrelevant effects. The results obtained are a further step towards implementing IIoT enabled safety critical wireless systems, as it has been possible to achieve levels of reliability very similar to those of a wired network.

Field testing of distributed functional safety systems is not always a feasible operation. For this reason, *Chapter* 5 deals with the implementation and calibration of a functional safety over wireless simulation model in OMNeT++. The calibration was done using measurements conducted in an experimental setup specifically designed for this purpose. The error channel model and polling time have been calibrated for various transmission channel conditions. The comparison with the experimental measurements and the tests conducted on a simulation with multiple nodes revealed a good calibration of the simulator.

However, further extensive tests are required for precise and complete validation.

Chapter 6 deals with creating a framework for the calculation and estimation of the Safety Function Response Time (SFRT). This metric is particularly incisive in the assessments of safety critical communication networks as it allows to estimate in the worst case what is the time to force the entire system to a fail-safe condition in case of communication errors or requests for execution of safety functions. In this chapter, we have proposed the modification of the SFRT equations defined in IEC 61784-3-3 for the specific use case with FSoE on Wi–Fi. Using these equations and the results of *Chapter* 5, it was possible to calculate the SFRT and verify, through the calibrated simulator, that these were comparable. Finally, the SFRT was evaluated in a real industrial use scenario that includes four mobile safety nodes. It has been shown how, even in the case of communication errors, the entire system can be forced to a safe state in a reasonable time, thus proving the feasibility, from a practical point of view, in using wireless networks in safety critical contexts.

Chapter 7 concludes the thesis with the assessment of different OPC-UA implementations for industrial IoT-based measurement applications. One of the concerns about OPC-UA is that its intrinsic high complexity can represent a bottleneck when used in low power distributed measurement systems. Various implementations of this protocol were then tested, and the CPU usage, cycle time, and power consumption on different operating system and communication network configurations were evaluated as key performance indexes. Generally speaking, it has been seen that the adoption of OPC-UA in distributed sensors is generally feasible both on wired and wireless networks. However, the use of the latter leads to an increase in cycle times and a decrease in determinism. This problem can be mitigated for both types of networks by using a combination of real–time operating systems and CPU isolation to reduce the disturbance effect of other competing processes to the measurement one. Positive results also emerged from the energy consumption point of view. Regardless of the communication network, it has been seen that the implementations based on compiled languages (C, C++) have a computational efficiency that potentially allows the use of battery-powered sensors even for long periods.

References

Abukwaik H., Gogolev A., Groß C., and Aleksy M. OPC UA Realization for simplified commissioning of adaptive sensing applications for the 5G IIoT. *Internet of Things*, 11:100221, September 2020. ISSN 25426605.

Adame T., Carrascosa M., and Bellalta B. Time-Sensitive Networking in IEEE 802.11be: On the Way to Low-latency WiFi 7. *arXiv:1912.06086 [cs]*, November 2020a.

Adame T., Carrascosa M., and Bellalta B. Time-Sensitive Networking in IEEE 802.11be: On the Way to Low-latency WiFi 7. *arXiv:1912.06086 [cs]*, November 2020b.

Åkerberg J., Reichenbach F., and Björkman M. Enabling safety-critical wireless communication using WirelessHART and PROFIsafe. In *2010 IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA 2010)*, pages 1–8, September 2010.

Åkerberg J., Gidlund M., Lennvall T., Neander J., and Björkman M. Efficient integration of secure and safety critical industrial wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2011(1):100, December 2011. ISSN 1687-1499.

Allouch A., Koubaa A., Khalgui M., and Abbes T. Qualitative and Quantitative Risk Analysis and Safety Assessment of Unmanned Aerial Vehicles Missions Over the Internet. *IEEE Access*, 7:53392–53410, 2019. ISSN 2169-3536.

Arm Architecture Reference Manual. Arm Architecture Reference Manual Armv8, for Armv8-A architecture profile. https://developer.arm.com/documentation/ddi0487/fb/.

Bertolotti I. C. and Hu T. Real-time performance of an open-source protocol stack for low-cost, embedded systems. In *ETFA2011*, pages 1–8, 2011.

Bianchi V., Boni A., Fortunati S., Giannetto M., Careri M., and De Munari I. A wi-fi cloud-based portable potentiostat for electrochemical biosensors. *IEEE Transactions on Instrumentation and Measurement*, 69(6):3232–3240, June 2020.

Bredel M. and Bergner M. On The Accuracy of IEEE 802.11g Wireless LAN Simulations Using OMNeT++. In *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*, Rome, Italy, 2009. ICST. ISBN 978-963-9799-45-5.

Breiling B., Dieber B., and Schartner P. Secure communication for the robot operating system. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–6, Montreal, QC, Canada, April 2017. IEEE. ISBN 978-1-5090-4623-2.

Bruckner D., Stanica M.-P., Blair R., Schriegel S., Kehrer S., Seewald M., and Sauter T. An Introduction to OPC UA TSN for Industrial Communication Systems. *Proceedings of the IEEE*, 107(6):1121–1131, June 2019. ISSN 0018-9219, 1558-2256.

Cavalaglio Camargo Molano J., Lahrache A., Rubini R., and Cocconcelli M. A new method for motion synchronization among multivendor's programmable controllers. *Measurement*, 126:202–214, October 2018.

Cavalcanti D., Bush S., Illouz M., Kronauer G., Regev A., and Venkatesan G. Wireless TSN–Definitions, use cases & standards roadmap. *Avnu Alliance*, pages 1–16, 2020.

Cavalcanti D., Perez-Ramirez J., Rashid M. M., Fang J., Galeev M., and Stanton K. B. Extending Accurate Time Distribution and Timeliness Capabilities Over the Air to Enable Future Wireless Industrial Automation Systems. *Proceedings of the IEEE*, 107(6):1132–1152, June 2019. ISSN 0018-9219, 1558-2256.

Chatzigiannakis I. and Tselios C. Internet of everything. In *Intelligent Computing for Interactive System Design: Statistics, Digital Signal Processing, and Machine Learning in Practice*, pages 21–56. 2021.

Chuanxiong G. and Shaoren Z. Analysis and evaluation of the TCP/IP protocol stack of Linux. In *WCC 2000-ICCT 2000. 2000 International Conference on Communication Technology Proceedings (Cat. No. 00EX420)*, volume 1, pages 444–453, 2000.

Damm M., Leitner S.-H., and Mahnke W. *OPC Unified Architecture.* Springer-Verlag Berlin Heidelberg, 2009.

Dawy Z., Saad W., Ghosh A., Andrews J. G., and Yaacoub E. Towards Massive Machine Type Cellular Communications. *arXiv:1512.03452 [cs, math]*, December 2015.

Ding L., Wang H., Xu A., and Li S. New considerations for SIL verification of functional safety fieldbus communication. *Journal of Loss Prevention in the Process Industries*, 43:488–502, September 2016. ISSN 09504230.

EN 50325-4. Industrial communication subsystem based on ISO 11898 (CAN) for controller-device interfaces. CANopen. European Committee for Standardization, 2002.

EN 50325-5. Industrial communications subsystem based on ISO 11898 (CAN) for controller-device interfaces. Functional safety communication based on EN 50325-4. European Committee for Standardization, 2010.

Espressif ESP8266. ESP8266 Wi-Fi MCU I Espressif Systems. `https://www.espressif.com/en/products/socs/esp8266`. (accessed 2021-09-28T14:19:55Z).

Etz D., Fruhwirth T., Ismail A., and Kastner W. Simplifying functional safety communication in modular, heterogeneous production lines. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4, Imperia, June 2018. IEEE. ISBN 978-1-5386-1066-4.

Etz D., Fruhwirth T., and Kastner W. Flexible Safety Systems for Smart Manufacturing. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1123–1126, Vienna, Austria, September 2020. IEEE. ISBN 978-1-72818-956-7.

Ferrari P., Flammini A., Rinaldi S., Sisinni E., Maffei D., and Malara M. Impact of quality of service on cloud based industrial IoT applications with OPC UA. *Electronicsweek*, 7(7):109, July 2018.

Frotzscher A., Wetzker U., Bauer M., Rentschler M., Beyer M., Elspass S., and Klessig H. Requirements and current solutions of wireless communication in industrial automation. In *2014 IEEE International Conference on Communications Workshops (ICC)*, pages 67–72, June 2014a.

Frotzscher A., Wetzker U., Bauer M., Rentschler M., Beyer M., Elspass S., and Klessig H. Requirements and current solutions of wireless communication in industrial automation. In *2014 IEEE International Conference on Communications Workshops (ICC)*, pages 67–72, Australia, June 2014b. IEEE. ISBN 978-1-4799-4640-2.

González I., Calderón A. J., Barragán A. J., and Andújar J. M. Integration of sensors, controllers and instruments using a novel OPC architecture. *Sensors*, 17(7), July 2017.

Grimaldi D. and Marinov M. Distributed measurement systems. *Measurement*, 30 (4):279–287, December 2001. ISSN 02632241.

Hadz, iaganovic A., Atiq M. K., Blazek T., Bernhard H.-P., and Springer A. The performance of openSAFETY protocol via IEEE 802.11 wireless communication. page 8.

Hashemian H. Aging management of instrumentation & control sensors in nuclear power plants. *Nuclear Engineering and Design*, 240(11):3781–3790, November 2010. ISSN 00295493.

Hashemian H. Wireless sensors for predictive maintenance of rotating equipment in research reactors. *Annals of Nuclear Energy*, 38(2-3):665–680, February 2011. ISSN 03064549.

Haskamp H., Meyer M., Möllmann R., Orth F., and Colombo A. W. Benchmarking of existing OPC UA implementations for Industrie 4.0-compliant digitalization solutions. In *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 589–594, 2017.

HMS Networks . Industrial network market shares 2020 according to HMS Networks. `https://www.hms-networks.com/news-and-insights/news-from-hms/2020/05/29/industrial-network-market-shares-2020-according-to-hms-networks`, 2020. (accessed 2021-05-05T13:53:43Z).

Howard J., Dighe S., Vangal S. R., Ruhl G., Borkar N., Jain S., Erraguntla V., Konow M., Riepen M., Gries M., Droege G., Lund-Larsen T., Steibl S., Borkar S., De V. K., and Van Der Wijngaart R. A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling. *IEEE Journal of Solid-State Circuits*, 46(1):173–183, January 2011. ISSN 0018-9200, 1558-173X.

IEC 61158-1. Industrial communication networks - Fieldbus specifications - Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series. International Electrotechnical Commission, 2019.

IEC 61508. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission, 2016.

IEC 61784-3. Industrial communication networks. Profiles. Part 3. International Electrotechnical Commission, 2021.

IEC 61784-3-12. Industrial communication networks - Profiles - Part 3-12: Functional safety fieldbuses - Additional specifications for CPF 12. International Electrotechnical Commission, 2010.

IEC 61784-3-3. Industrial communication networks - profiles - part 3-3: Functional safety fieldbuses - additional specifications for CPF 3. International Electrotechnical Commission, 2016.

IEC 61800-5-2. IEC 61800: Adjustable speed electrical power drive systems - Part 5-2: Safety requirements - Functional. International Electrotechnical Commission, 2016.

IEC 62541. OPC unified architecture - Part 1: Overview and concepts. International Electrotechnical Commission, 2016.

IEC 62734. Industrial networks - Wireless communication network and communication profiles - ISA 100.11a. International Electrotechnical Commission Sign in | Create account, 2014.

IEEE 802.11. IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, December 2016.

Ikram W., Jansson N., Harvei T., Fismen B., Svare J., Aakvaag N., Petersen S., and Carlsen S. Towards the development of a SIL compliant wireless hydrocarbon leakage detection system. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8, Cagliari, September 2013. IEEE. ISBN 978-1-4799-0864-6.

Islam K., Shen W., and Wang X. Wireless Sensor Network Reliability and Security in Factory Automation: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1243–1256, November 2012. ISSN 1094-6977, 1558-2442.

ISO 11898-1. Road vehicles - Controller area network (CAN). International Organization for Standardization, 2015.

Jesus Sobalvarro . Warehouse | 3D CAD Model Library | GrabCAD. https://grabcad.com/library/warehouse-6. (accessed 2021-07-05T07:36:25Z).

Karpowicz M. P. Energy-efficient CPU frequency control for the Linux system. *Concurrency and Computation: Practice and Experience*, 28(2):420–437, 2016. ISSN 1532-0634.

Khan M. O., Qiu L., Bhartia A., and Lin K. C.-J. Smart retransmission and rate adaptation in wifi. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pages 54–65. IEEE, 2015.

Kim J., Jo G., and Jeong J. A Novel CPPS Architecture Integrated with Centralized OPC UA server for 5G-based Smart Manufacturing. *Procedia Computer Science*, 155: 113–120, 2019. ISSN 18770509.

Lee H. and Ke K. Monitoring of Large-Area IoT Sensors Using a LoRa Wireless Mesh Network System: Design and Evaluation. *IEEE Transactions on Instrumentation and Measurement*, (9):2177–2187.

Lee K. B., Bernhard H.-P., Cavalcanti D., Pang Z., and Val I. Reliable, High-Performance Wireless Systems for Factory Automation. page 10, 2020.

Lee S., Kim C., and Lee J. Development of a smart sensor system using OPC UA. In *Proc. MoMM*, pages 220–225, Salzburg, Austria, 2017.

LeMaRiva|Tech. Raspberry Pi: The N-queens Problem! (benchmark) Preempt-RT vs. Standard Kernel! `https://lemariva.com/blog/2018/04/raspberry-pi-the-n-queens-problem-performance-test`. (accessed 2021-05-11T09:59:33Z).

Li Q., Tang Q., Chan I., Wei H., Pu Y., Jiang H., Li J., and Zhou J. Smart manufacturing standardization: Architectures, reference models and standards framework. *Computers in Industry*, 101:91–106, October 2018a. ISSN 01663615.

Li S., Xu L. D., and Zhao S. 5G Internet of Things: A survey. *Journal of Industrial Information Integration*, 10:1–9, June 2018b. ISSN 2452414X.

Liang C.-J. M., Chen K., Priyantha N. B., Liu J., and Zhao F. RushNet: Practical traffic prioritization for saturated wireless sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 105–118, Memphis Tennessee, November 2014. ACM. ISBN 978-1-4503-3143-2.

Liao Y. and Lai H. Investigation of a wireless real-time pH monitoring system based on ruthenium dioxide membrane pH sensor. *IEEE Transactions on Instrumentation and Measurement*, 69(2):479–487, February 2020.

Liggesmeyer P. and Trapp M. Safety in der industrie 4.0. In *Handbuch Industrie 4.0 Bd. 1*, pages 107–123. Springer, 2017.

Lo Bello L. and Steiner W. A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems. *Proceedings of the IEEE*, 107(6): 1094–1120, June 2019. ISSN 0018-9219, 1558-2256.

Lu Y. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6:1–10, June 2017. ISSN 2452-414X.

Maldonado R., Karstensen A., Pocovi G., Esswie A. A., Rosa C., Alanen O., Kasslin M., and Kolding T. Comparing Wi-Fi 6 and 5G Downlink Performance for Industrial IoT. *IEEE Access*, 9:86928–86937, 2021. ISSN 2169-3536.

Maniglia D. and Sofia R. *Context-Aware Edge Based Mechanisms to Improving IoT Data Transmission.* March 2019.

Meany T. Functional safety and Industrie 4.0. In *2017 28th Irish Signals and Systems Conference (ISSC)*, pages 1–7, Killarney, Co Kerry, Ireland, June 2017. IEEE. ISBN 978-1-5386-1046-6.

Montavon B., Peterek M., and Schmitt R. Model–based interfacing of large–scale metrology instruments. *Proc. SPIE 11059, Multimodal Sensing: Technologies and Applications, 110590C*, June 2019.

Morato A., Peserico G., Fedullo T., Tramarin F., and Vitturi S. Tuning of a simulation model for the assessment of Functional Safety over Wi-Fi. page 6.

Morato A., Vitturi S., Cenedese A., Fadel G., and Tramarin F. The Fail Safe over EtherCAT (FSoE) protocol implemented on the IEEE 802.11 WLAN. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1163–1170, Zaragoza, Spain, September 2019. IEEE. ISBN 978-1-72810-303-7.

Morato A., Vitturi S., Tramarin F., and Cenedese A. Assessment of Different OPC UA Industrial IoT solutions for Distributed Measurement Applications. In *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1–6, Dubrovnik, Croatia, May 2020. IEEE. ISBN 978-1-72814-460-3.

Morato A., Vitturi S., Tramarin F., and Cenedese A. Assessment of Different OPC UA Implementations for Industrial IoT-Based Measurement Applications. *IEEE Transactions on Instrumentation and Measurement*, 70:1–11, 2021. ISSN 0018-9456, 1557-9662.

Mordor Intelligence . Functional Safety Market | Growth, Trends, COVID-19 Impact, and Forecasts (2021 - 2026). https://www.mordorintelligence.com/industry-reports/functional-safety-market, 2021. (accessed 2021-05-06T13:26:02Z).

Munirathinam S. Industry 4.0: Industrial Internet of Things (IIOT). In *Advances in Computers*, volume 117, pages 129–164. Elsevier, 2020. ISBN 978-0-12-818756-2.

Omron . Omron Italia. https://omron.it/it/home. (accessed 2021-09-13T12:06:28Z).

Ooi B. and Shirmohammadi S. The potential of IoT for instrumentation and measurement. *IEEE Instrumentation and Measurement Magazine*, 23(3):21–26, May 2020.

Orfanus D., Indergaard R., Prytz G., and Wien T. EtherCAT-based platform for distributed control in high-performance industrial applications. In *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, September 2013.

Palisetty R. and Ray K. C. FPGA prototype and real time analysis of multiuser variable rate CI-GO-OFDMA. *IEEE Transactions on Instrumentation and Measurement*, 67(3):538–546, March 2018.

Palm F., Gruener S., Pfrommer J., Graube M., and Urbas L. Open source as enabler for OPC UA in industrial automation. In *Proc. ETFA*, pages 1–6, Luxembourg, Luxembourg, 2015.

Peserico G., Fedullo T., Morato A., Tramarin F., Rovati L., and Vitturi S. SNR-based Reinforcement Learning Rate Adaptation for Time Critical Wi-Fi Networks: Assessment through a Calibrated Simulator. In *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1–6, Glasgow, United Kingdom, May 2021a. IEEE. ISBN 978-1-72819-539-1.

Peserico G., Fedullo T., Morato A., Tramarin F., and Vitturi S. Wi–Fi based Functional Safety: An Assessment of the Fail Safe over EtherCAT (FSoE) protocol. page 8, 2021b.

Peserico G., Morato A., Tramarin F., and Vitturi S. Functional Safety Networks and Protocols in the Industrial Internet of Things Era. *Sensors*, 21(18):6073, January 2021c.

Pfrommer J. Semantic interoperability at big-data scale with the open62541 OPC UA implementation. In *Proc. Workshop Interoperability and Open-Source Solutions for the Internet of Things*, pages 173–185, Stuttgart, Germany, 2016.

Pimentel V. and Nickerson B. G. A safety function response time model for wireless industrial control. In *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, pages 3878–3884, Dallas, TX, USA, October 2014. IEEE. ISBN 978-1-4799-4032-5.

Prytz G. A performance analysis of EtherCAT and PROFINET IRT. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 408–415. IEEE, September 2008.

Raspberry Pi. Raspberry Pi Foundation. https://www.raspberrypi.org/. (accessed 2021-05-18T12:02:03Z).

Refaat T. K., Daoud R. M., Amer H. H., and Makled E. A. WiFi implementation of wireless networked control systems. In *2010 Seventh International Conference on Networked Sensing Systems (INSS)*, pages 145–148, June 2010a.

Refaat T. K., Daoud R. M., Amer H. H., and Makled E. A. WiFi implementation of Wireless Networked Control Systems. In *2010 Seventh International Conference on Networked Sensing Systems (INSS)*, pages 145–148, Kassel, Germany, June 2010b. IEEE. ISBN 978-1-4244-7911-5.

Rizzi M., Ferrari P., Flammini A., and Sisinni E. Evaluation of the IoT LoRaWAN solution for distributed measurement applications. *IEEE Transactions on Instrumentation and Measurement*, 66(12):3340–3349, December 2017.

Robert J., Georges J.-P., Rondeau E., and Divoux T. Minimum cycle time analysis of Ethernet–based real-time protocols. *International Journal of Computers, Communications and Control*, 7(4):743–757, 2012.

Robinson S. Living with the Challenges to Functional Safety in the Industrial Internet of Things. In *Living in the Internet of Things (IoT 2019)*, pages 35 (6 pp.)–35 (6 pp.), London, UK, 2019. Institution of Engineering and Technology. ISBN 978-1-83953-089-0.

Seno L., Tramarin F., and Vitturi S. Performance of Industrial Communication Systems: Real Application Contexts. *IEEE Industrial Electronics Magazine*, 6(2): 27–37, June 2012. ISSN 1932-4529.

Sestito G. S., Turcato A. C., Dias A. L., Rocha M. S., da Silva M. M., Ferrari P., and Brandao D. A Method for Anomalies Detection in Real-Time Ethernet Data Traffic Applied to PROFINET. *IEEE Transactions on Industrial Informatics*, 14 (5):2171–2180, May 2018. ISSN 1551-3203, 1941-0050.

Sha K., Shi W., and Watkins O. Using Wireless Sensor Networks for Fire Rescue Applications: Requirements and Challenges. In *2006 IEEE International Conference on Electro/Information Technology*, pages 239–244, East Lansing, MI, USA, May 2006. IEEE. ISBN 978-0-7803-9592-3 978-0-7803-9593-0.

Shen W., Zhang T., Barac F., and Gidlund M. PriorityMAC: A Priority-Enhanced MAC Protocol for Critical Traffic in Industrial Wireless Sensor and Actuator Networks. *IEEE Transactions on Industrial Informatics*, 10(1):824–835, February 2014. ISSN 1551-3203, 1941-0050.

Sisinni E., Saifullah A., Han S., Jennehag U., and Gidlund M. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Trans. Industr. Inform.*, 14(11):4724–4734, November 2018.

Skrzypczak L., Grimaldi D., and Rak R. Analysis of the different wireless transmission technologies in Distributed Measurement Systems. In *Proc. IDAACS*, pages 673–678, Rende, Italy, 2009.

Sudhakaran S., Mageshkumar V., Baxi A., and Cavalcanti D. Enabling QoS for Collaborative Robotics Applications with Wireless TSN. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, Montreal, QC, Canada, June 2021. IEEE. ISBN 978-1-72819-441-7.

Sudhakaran S., Montgomery K., Kashef M., Cavalcanti D., and Candell R. Wireless Time Sensitive Networking for Industrial Collaborative Robotic Workcells. page 4.

Taylor J. H., Akerberg J., Ibrahim H. M. S., and Gidlund M. Safe and secure wireless networked control systems. In *2012 IEEE International Conference on Control Applications*, pages 871–878, Dubrovnik, Croatia, October 2012. IEEE. ISBN 978-1-4673-4505-7 978-1-4673-4503-3 978-1-4673-4504-0.

Tian G. Y. Design and implementation of distributed measurement systems using fieldbus-based intelligent sensors. *IEEE Transactions on Instrumentation and Measurement*, 50(5):1197–1202, October 2001.

Tramarin F., Mok A. K., and Han S. Real-Time and Reliable Industrial Control Over Wireless LANs: Algorithms, Protocols, and Future Directions. *Proceedings of the IEEE*, 107(6):1027–1052, June 2019. ISSN 0018-9219, 1558-2256.

Tramarin F., Vitturi S., Luvisotto M., and Zanella A. On the Use of IEEE 802.11n for Industrial Communications. *IEEE Transactions on Industrial Informatics*, 12(5):1877–1886, October 2016. ISSN 1551-3203, 1941-0050.

U L. H., Yang J., Cai Y., Karlapalem K., Liu A., and Huang X., editors. *Web Information Systems Engineering: WISE 2019 Workshop, Demo, and Tutorial, Hong Kong and Macau, China, January 19–22, 2020, Revised Selected Papers*, volume 1155 of *Communications in Computer and Information Science*. Springer Singapore, Singapore, 2020. ISBN 9789811532801 9789811532818.

Unified Automation. UA ANSI C Server Professional: OPC UA Subscription Concept. http://documentation.unified-automation.com/uasdkc/1.4.2/html/L2UaSubscription.html. (accessed 2021-06-08T13:03:54Z).

van Kranenburg R. and Dodson S. *The Internet of Things: A Critique of Ambient Technology and the All-Seeing Network of RFID*. Network Notebooks. Institute of Network Cultures, 2008. ISBN 978-90-78146-06-3.

Vitturi S. Stochastic model of the Profibus DP cycle time. *IEE Proceedings - Science, Measurement and Technology*, 151(5):335–342, September 2004. ISSN 1350-2344, 1359-7094.

Vitturi S., Carreras I., Miorandi D., Schenato L., and Sona A. Experimental Evaluation of an Industrial Application Layer Protocol Over Wireless Systems. *IEEE Transactions on Industrial Informatics*, 3(4):275–288, November 2007. ISSN 1551-3203.

Vitturi S., Morato A., Cenedese A., Fadel G., Tramarin F., and Fantinel R. An innovative algorithmic safety strategy for networked electrical drive systems. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pages 368–373, Porto, Portugal, 2018. IEEE.

Vitturi S., Zunino C., and Sauter T. Industrial Communication Systems and Their Future Challenges: Next-Generation Ethernet, IIoT, and 5G. *Proceedings of the IEEE*, 107(6):944–961, June 2019. ISSN 0018-9219, 1558-2256.

Weyer S., Schmitt M., Ohmer M., and Gorecky D. Towards Industry 4.0 - Standardization as the crucial challenge for highly modular, multi-vendor production systems. *IFAC-PapersOnLine*, 48(3):579–584, 2015. ISSN 24058963.

Wijethilaka S. and Liyanage M. Survey on Network Slicing for Internet of Things Realization in 5G Networks. *IEEE Communications Surveys & Tutorials*, 23(2): 957–994, 2021. ISSN 1553-877X, 2373-745X.

Willig A. and Wolisz A. Ring stability of the PROFIBUS token-passing protocol over error-prone links. *IEEE Transactions on Industrial Electronics*, 48(5):1025–1033, 2001. ISSN 02780046.

WirelessHART . *HART Field Communication Protocol Specification, Rev. 7.7.* 2020.

Witten I. H., Frank E., and Hall M. A. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, Boston, third edition, 2011. ISBN 978-0-12-374856-0.

Wollschlaeger M., Sauter T., and Jasperneite J. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Industrial Electronics Magazine*, 11(1):17–27, March 2017. ISSN 1932-4529.

Xie G., Li Y., Han Y., Xie Y., Zeng G., and Li R. Recent Advances and Future Trends for Automotive Functional Safety Design Methodologies. *IEEE Transactions on Industrial Informatics*, 16(9):5629–5642, September 2020. ISSN 1551-3203, 1941-0050.

Xu Y., Sun Y., Liu Y., Wang Y., Gu P., and Liu Z., editors. *Nuclear Power Plants: Innovative Technologies for Instrumentation and Control Systems: The Fourth International Symposium on Software Reliability, Industrial Safety, Cyber Security and Physical Protection of Nuclear Power Plant (ISNPP)*, volume 595 of *Lecture Notes in Electrical Engineering.* Springer Singapore, Singapore, 2020. ISBN 9789811518751 9789811518768.

Yang D., Ma J., Xu Y., and Gidlund M. Safe-WirelessHART: A Novel Framework Enabling Safety-Critical Applications Over Industrial WSNs. *IEEE Transactions on Industrial Informatics*, 14(8):3513–3523, August 2018. ISSN 1551-3203, 1941-0050.

Yavari A., Jayaraman P. P., Georgakopoulos D., and Nepal S. ConTaaS: An Approach to Internet-Scale Contextualisation for Developing Efficient Internet of Things Applications. In *Hawaii International Conference on System Sciences*, 2017.

Zheng T., Gidlund M., and Akerberg J. WirArb: A New MAC Protocol for Time Critical Industrial Wireless Sensor Network Applications. *IEEE Sensors Journal*, 16 (7):2127–2139, April 2016. ISSN 1530-437X, 1558-1748, 2379-9153.