


Doctoral Dissertation
Doctoral Program in Control and Computer Engineering (33^{th} cycle)

Industry 4.0: Industrial IoT Enhancement and WSN Performance Analysis

Mohammad Ghazi Vakili

* * * * * *

Supervisor
Prof. Claudio Giovanni Demartini

Doctoral Examination Committee:
Prof. Paulo Portugal, Referee, University of Porto (PT)
Prof. Davide Quaglia, Referee, University of Verona (IT)
Dr. Luca Durante, Senior Researcher at CNRI-EIIT (IT)
Prof. Bartolomeo Montrucchio, Politecnico di Torino (IT)
Prof. Lucia Lo Bello, University of Catania (IT)

Politecnico di Torino
September 2021

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Mohammad Ghazi Vakili

Turin, September 2021

Summary

The fourth revolution of industry began in 2011 at Hannover Fair, with the introduction of communication between the factory shop floor and information technology applications. The Internet of Things (IoT) technology leveraged this revolution to transfer data from industrial plants to cloud services, providing manufacturing improvements, such as production line optimization (e.g., reducing production costs) and machine learning (e.g., suggesting the best product-to-produce in the next month). This new production era provided a new customer experience by customizing the product for the customer. Thus, it developed a new business model for the business owners as well as new customer experiences. This new methodology was coined Industry 4.0, and it requires sustainable communication, intra- and inter-Shopfloor, and Business devices and applications.

There are many communication challenges in Industry 4.0; however, the present work focuses on IoT and Wireless Sensor Network (WSN). The first part of this thesis deals with IoT issues in industrial applications, and the second part proposes a mathematical model to analyze the 6TiSCH WSN performance indicators for industrial applications. Moreover, the TSCH predictor is presented to simulate and predict realistic performance indicators in the TSCH network in the WSN domain. In general, three proposals focus on the Industry 4.0 communication challenges.

The first part of this thesis is focused on IoT standardization issues, and it proposes an OPC-IoT protocol to overcome standardization challenges. The proposed IoT platform's communication is based on the Reference Architecture Model Industries 4.0 (RAMI 4.0), which means it complies with the OPC-UA protocol as the RAMI 4.0 suggests. Nonetheless, the platform proposes an industrial gateway that supports industrial protocols, such as Profinet S7, Modbus, and OPC-UA. Experimental analysis was performed to compare the OPC-IoT platform to the commercial Kaa IoT platform.

The IoT domain's other challenges are centralization and decentralization of the data and control logic in the factories. These challenges are related to the architectural issues in Industry 4.0. This issue mainly affects data latency and data privacy. The fog architectures are proposed to overcome centralization challenges in industrial applications. IFog4.0 was developed and implemented for industrial applications in Industry 4.0, which is compliant with the RAMI 4.0 and utilizes many

open-source components. In this thesis, the industrial use case was implemented by using IFog4.0.

The second part of the thesis focuses on Industrial Wireless Sensor Networks, precisely analyzing the Time Slot Channel Hopping (TSCH) technique introduced by the IEEE 802.15.4 standard for WSN. The analysis was performed mainly on an 6TiSCH enabled device, IPv6, over the TSCH network. Communication performance in wireless sensor networks suffers from background traffic such as Wi-Fi or other networks that operate in the same spectrum. However, thanks to its ability to effectively counteract disturbances and interference, including the traffic generated by co-located Wi-Fi networks, TSCH is currently gaining momentum in many application fields characterized by demanding reliability determinism requirements. In particular, the ability of TSCH to sensibly change the transmission frequency on every attempt sensibly mitigates packet loss, improving the overall behavior tangibly.

The challenge with the WSN is how to select an exemplary configuration in the WSN, especially in the 6TiSCH protocol, which is the latest version of the TSCH protocol. A mathematical model is proposed to overcome the WSN network configuration issue by analyzing the TSCH's behavior and propose a model to estimate the performance indicator in the 6TiSCH devices. To better analyze the TSCH WSN's behavior, the communication quality obtained by the 6TiSCH protocol in a setup that includes real WSN devices exposed to reality is evaluated experimentally. A theoretical model is then developed, based on simple assumptions about time and the effectiveness of frequency diversity, which satisfactorily matches the actual behavior. The model permits the determination of the number of network parameters(e.g., the retry limit) that actually affect communication quality and can be exploited to find proper settings for them. Finally, the ability of channel hopping to prevent narrowband interference from disrupting communication is assessed. As the results show, this mechanism makes motes suffer from an equivalent interference that roughly corresponds to the mean interference evaluated over all physical channels.

In addition to the proposed mathematical model, the accurate measurement has been performed on the OpenMote B devices to evaluate the power consumption value. These experiments provide the actual energy consumption for each slot frame on the devices. The values are utilized to develop a realistic power-consumption model, which, to the best of our knowledge, is the first realistic model for OpenMote B .

Additionally, the proposed model provides a reasonable estimation of the performance indicators in the 6TiSCH WSN(e.g., reliability, power consumption, and latency). The single and multi-hop WSN mathematical models were compared with 6TiSCH's behaviors on the actual device and were utilized to predict the network behavior where the parameters of the 6TiSCH matrix are varied. The reason to propose such a model was to verify the performance requested from the industry

and satisfy different application requirements in the factory. The results demonstrated that when one of the three performance indicators is privileged, the others worsened. These results open a new research direction for developing a model to calculate 6TiSCH parameters with reinforcement learning for the WSN network. Eventually, the proposed parameters could satisfy the application requirements after applying the learning technique.

The proposed mathematical model is applicable when there is no packet waiting for the next hop in the multi-hop network and develops the proposed model for the TSCH network; however, this assumption is not a limitation in many WSN applications due to the low frequency of transferring data in the WSN devices. Nonetheless, it could be useful to propose a technique or tool to overcome this issue. The TSCH predictor is a simulation tool proposed for predicting the performance indicators in the TSCH WSN, and it overcomes the queuing phenomenon in the WSN. This predictor simulates the TSCH devices with excellent estimation, then provides the performance indicators. Experimental analysis was performed on real OpenMote B devices to validate the TSCH Predictor results.

In IoT and WSN domains, we explore the strengths and weaknesses of communication technologies concerning standardization and performance indicators, especially their potential to serve real-world applications. The proposed methods provide a new generation of easy-to-deploy platforms in Industry 4.0. However, by applying communication models in WSN, we demonstrate both potential use cases and the concerns that may limit them. By revealing problems that future deployments may face, we hope to guide improvements to these protocols that will improve their use and support the growth of the Internet of Things and Wireless Sensor Networks in industrial environments for the future of factories.

Acknowledgements

Many helped me along the way on this journey. I want to take a moment to thank them.

I would like to thank my supervisor Prof. Claudio Demartini for his invaluable advice, continuous support, and patience during my Ph.D. journey. His knowledge and plentiful experience have encouraged me in my academic research and daily life.

My gratitude extends to the Department of Control and Computer and Doctorate school for the funding opportunity to undertake my Ph.D. studies.

Additionally, I would like to thank Prof. Bartolomeo Montrucchio for his treasured support, which was influential in shaping my experiences during my Ph.D. and personal life. Additionally, I also thank Dr. Stefano Scanzio and Dr. Gianluca Cena for their mentorship and invaluable advice and support in shaping my experiment methods and critiquing my results, and helped me step by step to reach to my goals.

I wish to thank my dissertation referee and committee, Prof. Paulo Portugal and Prof. Davide Quaglia, who served as wise referee members, and Prof. Lucia Lo Bello and Dr. Luca Durante, served as a committee members.

I would like to thank Edoardo for being my best friend and colleague, who helped me during my hard times. Also, I would like to thank my best friend, Mauro, who helped me sincerely all the time.

I would like to take a moment to thank all my lab mates, colleagues, and research team – Renato, Fillipo, Francesco, Sina, Davide, Gabriele, Antonio, Federico, Gustavo, Sorath, and Alberto for a cherished time spent together in the lab, and in social time. My appreciation also goes out to my family and friends for their encouragement and support throughout my studies.

Finally, to my wife, Elahe: your love and understanding helped me through the dark times. Without you believing in me, I never would have made it. It is time to celebrate; you earned this degree right along with me.

I dedicate this thesis to my wife, for her constant support and unconditional love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The term **Internet of Things (IoT)** was coined in 1999 by Kevin Ashton, who used Radio Frequency IDentification (RFID) [1] to read sensor data. Over the past 20 years, IoT technology has evolved and expanded its communication capabilities with the use of Wi-Fi , wireless sensor network (WSN), 3G, 4G, and 5G, and these technologies help increase connectivity around the world. For instance, 5G technology made it possible for IoT to provide real-time connectivity, which is essential in critical industrial applications [2]. IoT technology enables various services around the world, from smart homes to smart factories.

The integration of Internet technology into the industry has been committed to the birth of Industry 4.0, which suggested approaches and guidelines call for a more significant attempt to combine all current technologies into full and efficient industrial goods [3]. New specifications have been adopted to unify hardware tools, manufacturing machinery, and applications, leading to a linked world in which all parties connect, share information, and monitor manufacturing activities [4].

The term of "Industry 4.0" was introduced initially in 2011, at Hanover Fair, where a cyber-physical system (CPS) made a connection between the physical and digital world. The CPS enabled real-time control and monitoring of physical processes by providing sensor data. It proposed physical-layer visualization and developed a new technology called "digital twin" using physical processes [5]. It also helped develop a flexible, adaptable manufacturing system through its integration with IoT and WSN technologies [6]. IoT technology could develop smart production by adding intelligent machines and processes to factory floors. This innovation could be adopted during production and subsequent phases, including the continuous monitoring of product lifecycles.

The ongoing industrial revolution of the past decade is taking enormous advantage of cloud computing, as cloud computing improves product quality, efficiency, and decision-making [7]. The fourth industrial revolution, is expanding based on the connection between industry and the Internet [8]–[10]. The industrial world is moving toward the concept of connected *things*, in which machines are linked

through the cloud, interacting with a complex virtual world. Cloud computation enables services that were not previously available, such as the machine learning service, which develops smarter devices by computing large amounts of data on the cloud side.

On the other hand, Wireless sensor networks (WSN) is another key technology in industry 4.0, were introduced about two decades ago to provide accessible and affordable sensors with low-cost communication over the air [11]. WSNs aim to collect data through wireless communication from many sensors deployed across large areas, and they are becoming key technologies for the IoT, allowing data exchange between sensors and edge devices, such as gateways and routers. WSNs are basic instruments that can quickly be inserted into a monitoring or control scheme, sometimes operated by batteries, with on-board sensors and, often, actuators. When the wireless sensor and actuator networks are embedded together, they are referred to as wireless sensor and actor networks (WSAN).

WSN nodes usually do not directly connect to the Internet; typically, the root has an Internet connection, making a difference between WSN and IoT devices in the architectural schema. WSNs are used in large areas, and sensor networks, and this technology could leverage the accessibility of data and easily communicate between devices. This technology could deploy in many industrial applications, such as natural-disaster management [12], [13], environmental monitoring [14], [15], precision agriculture [16], [17], unmanned surveillance [18], assistance for maintenance in large industrial plants [19], [20], smart cities [21], and wireless body area networks (WBAN) for healthcare [22].

The present work focuses on performance indicators in the IoT and Wireless Sensor Network (WSN) technologies for industry 4.0. This dissertation is structured as follows: the first part deals with IoT issues in industrial applications. Chapter 2 and 3 proposes Industrial Internet of things platform for industry 4.0. Then the second part proposes mathematical models to analyze the 6TiSCH WSN performance indicators for industrial applications. In the following, chapter 4 explains the testbed configuration and experimental setup. Chapter 5 and 6 propose mathematical models to analyze the performance indicators. Chapter 7 presents a TSCH predictor to simulate and predict performance indicators in the TSCH network for WSN domain. In general, three proposal deals with communication challenges in the Industry 4.0.

1.1 Internet of things: Problem Statement and Challenges

Most industrial communication protocols were developed before the advent of the Internet, and the first industrial network, which was based on serial communication, was proposed by Modcon in 1979. Other standards were introduced by other

companies, such as Bitbus, Hart, and DeviceNet, and, after 2000, other standards were developed using Ethernet as the base technology [2].

Industry 4.0 aims to apply IoT technology in factories to connect machines and assets to remote servers. This connection will result in an intelligent system that relies on cloud computing [23], [24]. However, common IoT technology is not based on industrial protocols, so gateways and middleware are acting as important rules to enable communication between machines and intelligent systems hosted on the cloud.

The Reference Architecture Model Industries 4.0 (RAMI 4.0) was introduced by Platform Industries 4.0 [25], [26], and was developed to create a road map for Industry 4.0 standardization, especially in the Industry 4.0 integration. It consists of a three-dimensional map that explains the structural organization of Industry 4.0 [27]. RAMI 4.0 is a multidisciplinary standard that helps enterprises adapt to the aspects of Industry 4.0. It is a collection of existing standards divided into four categories-architecture, life cycle, value stream, and hierarchical levels-and explains how to integrate them into each factory layer. The standards community is also developing new standards to help companies fill the standards gap between information and communications technology (ICT) and industry needs [25].

The Industry 4.0 integration challenges are categorized by two problems in the CPS layer: the IoT protocol standardization and centralized and decentralized issues, such as latency and data privacy. These two problems lead us to propose standard Industrial Internet of things (IIoT) protocols based on OPC-UA and the fog architecture for Industry 4.0.

1.1.1 IoT Challenges

IoT standardization protocol

Industry 4.0 improved the automation of manufacturing by adopting IoT in factories. As a result, the factories became smarter and predictable by implementing communication between assets and business applications [28]. CPSs in the industry are based on defined standards established by either the international standard or a local standardization institute. These standards have always been a requirement in industrial design; however, since 1999, many IoT protocols have been proposed to exchange data between connected things. These protocols-some of which are MQTT, AMQP, XMPP, DDS, Websocket, and restful services [29]-have become a backbone for IoT platforms. Thus, standardization of IIoT platforms is essential.

IoT gateways also are proposed for machine-to-machine (M2M) communication and data marshaling in the IoT scenario to handle IoT end nodes. Gateways are used as part of the edge computing paradigm in IoT architectures [30], [31]. The industrial gateway is customized for industrial networks that comply with industrial protocols and assets, such as PLC and instruments. The gateways are developed to

connect industrial networks to IoT platforms, and they work as a data-conversion layer [32]. The data conversion gateway accepts different input protocols, such as *Profinet*, *Modbus*, and *Hart*, and then the output of the gateway provides an IoT protocol that exchanges data from the gateway to the IoT platform/middleware.

As mentioned previously, RAMI 4.0 is becoming the standard architecture model for Industry 4.0. VDE has proposed its architecture and suggested RAMI 4.0 as a standard architecture for Industry 4.0 [24]. In RAMI 4.0, the data conversions are placed in a communication and information layer. As a result, the model places an IEC 62541 standard in the communication layer, and it proposes the OPC-UA protocol as a suggested protocol for Industry 4.0 [33], which complies with the IEC 62541 standard. Furthermore, AutomationML is proposed for an end-to-end engineering approach in industrial scenarios [34], [35].

Chapter 2 proposes an Industry 4.0 platform that can satisfy the industrial standard requirements. The challenges are standardization in the communication layer and proposing a unique protocol to exchange data from machines to the cloud system. An IoT middleware architecture is proposed and developed to connect directly to the machine protocol. The proposed solution complies with RAMI 4.0 and develops a platform for industrial needs.

Centralization in industry

The Industry 4.0 development uses cloud computing as an intelligent system that controls machines and robots remotely. The centralized cloud services are used for cloud computing. Recently, the modern cloud-computing model has been adopted to overcome the need for centralized assessment and a centralized storage unit, which has partially resolved the digital transformation problems relevant to the IoT environment [36]. However, network capabilities have grown with speed disproportionate to the growth of computing capacity. Thus, the cloud-based infrastructure cannot have the bandwidth needed to run at the speed required to satisfy the crucial real-time limits [2].

One of the most common techniques is to deploy sensitive tasks on nodes near the end-user application. This model, best known as *fog computing* [37], leaves a centralized cloud computing concept heading in a decentralized architecture, where edge nodes perform essential tasks. This paradigm decreases latency in time-critical applications and leaves regular computing to the central cloud service [38].

The decentralization of computation introduces a new issue regarding the deployment of all cloud/fog services. Security is negatively affected by offloading computation to edge nodes [39], and it becomes more complicated to run different services across the network without a simple and effective way of deploying modules to nodes that perform some computation [40], [41]. Moreover, most available solutions are proprietary and require fees or subscriptions to be used, reducing the possibility of creating products tailored to the user's needs.

Thus, Chapter 3 proposes an *open-source* architecture for IIoT, to evaluate if open-source tools can provide the same solution as proprietary software. The proposed architecture is specifically designed for small- to medium-sized enterprises (SME). Further, a case study is introduced to explore the usefulness of the implementation process.

1.1.2 Related work

IoT platforms and middleware are key technologies in Industry 4.0. These technologies have been developed for several applications, from smart cities to industrial applications. Industry 4.0 is a road map that is leveraged with many different technologies. In this section, we will present some of those key elements.

Industry 4.0

The standard definition of Industry 4.0 began by combining IoT and CPS communications at the factory level. This hybrid technology helped connect machine data to machine data and machine data to human data in real-time. *Connected manufacturing*, a new concept in the industrial domain, connects factories and organizations by using Internet of Service (IoS) technology [42].

Recently, emerging technologies and facilities have also made it easier to develop intelligent and scalable factories that provide customized products and produce on-demand products. As a result, Industry 4.0 is now an economic imperative to decrease prices, raise manufacturing income, and minimize operating costs [43]–[47].

RAMI 4.0

Working in the industrial domain requires using relevant standards for each sector, making Industry 4.0 communication different concerning other domains. Therefore, the (RAMI 4.0) was proposed by VDI/VDE-GMA 7.21 Industrie 4.0 technical committee for Industry 4.0 standardization. This model defines data exchange methods, from assets to the business application layer [25]. Furthermore, RAMI 4.0 introduces many standards for communications and information in the CPS that improve communication from assets to the information layer. RAMI 4.0 suggests the OPC-UA protocol (an M2M protocol based on IEC62541) as a preferred protocol in the communication layer. The OPC-UA was developed by the OPC foundation [1], which complies with the IEC 62541 standard [33].

RAMI 4.0 represents a set of integration standards to improve the Industry 4.0 implementation [25]. Fig. 1.1 presents RAMI 4.0 as a service-oriented architecture

[1] https://opcfoundation.org/

in the three-axis view. The first axis represents the hierarchy from a network model to a hardware-based structure; the second axis explains the products' life cycles, from development to usage and maintenance; and the third axis represents the architecture of the layered service. Security is at the base of the RAMI 4.0 model and the applications implemented by this model are secure by design [48].

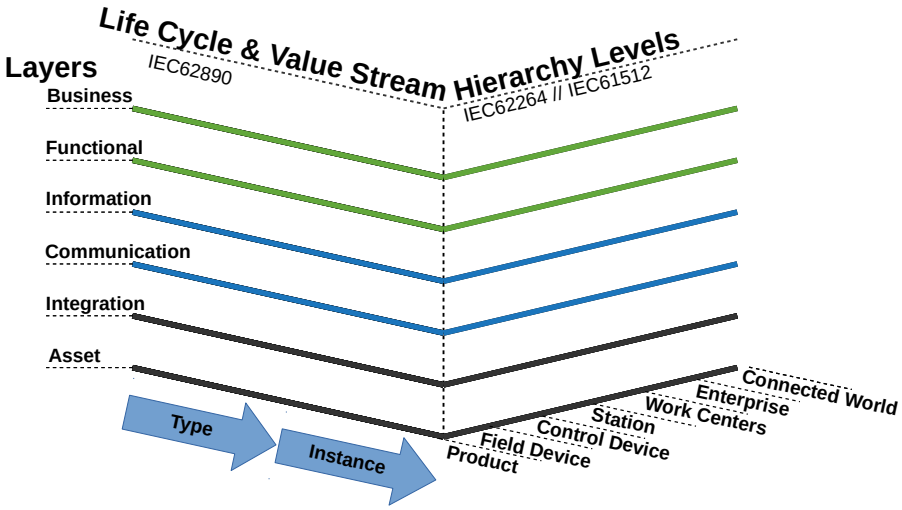


Figure 1.1: RAMI 4.0 architecture

IoT middleware platforms

Middleware provides a software interface between applications, the operating system (OS), and network communication to reduce the complexity of the system [49]. In general, the middleware's task is to communicate with assets in a different system's layer in the IoT architecture and manage the data exchange between sensors and IoT storage [49]. Middleware is the best solution for managing multiple assets and protocols in the CPS, and it is used mainly in the Industry 4.0 communication layer.

As reported in [50], several challenges must be addressed before developing middleware, such as inseparability, scalability, spontaneous interaction of objects and devices, and infrastructure diversity. The middleware functionality can be classified as service-based, event-based, VM-based, tuple-spaces, database-oriented, and application-specific [49].

Industrial middleware is expected to fulfill industrial requirements, such as low latency, real-time communication (either soft or hard), and working with international standards. These specifications are necessary for ensuring reliable and rapid data exchange between assets and business applications. Moreover, industrial middleware needs to provide SDKs for further development.

IoT platforms

An IoT platform is a set of functionalities that builds an ecosystem in which things (sensors, actuators, and APPs) can connect and communicate. The IoT platform includes middleware too, and the platform provides many functionalities, such as communication, databases for big data, the representational state transfer (REST) API, user management / end-node management, flexibility to implement data schema, event/alert management, visualization, and SDKs for development [51].

In Industry 4.0, IoT platforms are used in the communication layer to exchange information, and they play a crucial role in the factory's IT infrastructure. Essential data rely on IoT platforms, so using the right technology is critical. The first step to choosing the right IoT technology is to identify problems by considering the industrial scenario; then, it is possible to develop an IoT solution considering the requirements [51].

Several metrics are defined to evaluate an IoT middleware platform, which may be defined as two subcategories: qualitative and quantitative. Qualitative metrics are popularity, number of updates, security features, supports, and the SDK for development. The quantitative metrics are error percentage, fairness, packet size, and latency of the server application. IoT platforms are usually evaluated using these metrics [51].

RAMI 4.0 suggests several standards adopt Indutry4.0; this is why Industrial IoT platforms currently face additional challenges related to adapting the communication protocol from assets to the business layer in Industry 4.0.

The most popular platforms and IoT middleware are Eclipse Kura, ThingsWorx, Kaa, Devicehive, Thinger.io, ThingSpeak, OpenIoT, Linksmart, Bosch IoT, Webinos, Samsung ARTIK, Webinos, Nimbits, Konker Platform, Orion, Nitrogen, and Sitewhere. SDKs for development are provided in some of these (Kaa, Eclipse Kura, and ThingsWorx). After review, the Kaa platform was selected to compare with the proposed protocol, as the Kaa IoT was flexible and compatible with different databases. [51].

Gateways

Industrial IoT architecture utilizes gateways to perform data marshaling. It is common to use many vendors in factory communication, each of which has its technologies and protocols. However, developing human-machine interfaces for Industry 4.0 requires reliable communication and data exchange between different machines, such as robots, CNCs, and 3D printers. These requirements are why data marshaling needs to be provided in the Industrial IoT scenario to convert the machine data format to a format that is acceptable for IoT middleware. IoT middleware offers gateways, solves compatibility issues, and allows for exchanging data using M2M communication [52].

Fog

Improvements in cloud computation have been enabled by the great effort of researchers in recent years, as well as the evolution of the technology on which cloud computing is based. The use of a centralized architecture, however, limits the real-time application of this paradigm, as network technology is not experiencing the same revolution as parallel and cloud computation. A solution that has been recently proposed is to move some of the computational power of the central nodes toward the edges of the network (near end-user nodes). This approach has been called *fog computing* [53].

Decentralization allows for creating specialized nodes dedicated to real-time computation. These nodes perform tasks that would be negatively affected by the physical distance from cloud server clusters or by network capabilities. The tasks that manage sensitive or harmful activity are required on leaf nodes that provide a quicker response and do not violate safety criteria. Services for which latency is not critical may still be executed on the central server cluster [41].

Fog computing proposes a new computational framework for the industrial environment, especially Industry 4.0. The fog architecture extends the cloud approach to highly critical tasks. However, the decentralization of computational power increases the complexity of the industrial framework, fragmenting the execution of services across the node network. Therefore, it is essential to provide an architecture capable of running all essential services seamlessly, regardless of physical location. A fast service deployment environment can dramatically increase the productivity and quality of industrial frameworks [54].

Databases

The IIoT platform requires a reliable, fast, and robust data-storage system. This is why nonrelational databases (DB) are selected in the IoT scenario, which enables fast storage operation with low computational power. Several NoSQL DB has recently been developed, divided into four categories: Wide-Column Store, Graph Store, Key-Value Store, and Document Store. *Graph Store* is developed based on graph theory, and graph connections represent each relationship. *Key-Value Stores* is implemented for storing data in a schema-less way, where each data is stored as a pair of index values. Dynamic DB, Cassandra, and Berkeley are implemented by storing the key-value schema. The *Wide-Column Stores* were developed based on Google®Bigtable. The data structure is a tabular form of columns and rows. HyperBase, BigTable[2], and HBase are Column Store databases [55], [56]. The *Document database* is an extended version of the key-value schema, where each key is a standard document, such as JSON, BSON, or XML. They have a flexible schema

[2]https://cloud.google.com/bigtable/

and can be modified as needed. Couchbase Server and MongoDB[3] are document databases. These databases are like collection-based document technology [55].

Furthermore, the streaming data platform is an important feature in the distributed IoT platform. This kind of platform is like a database in IoT platforms for streaming data in real-time. Kafka is open-source and widely used in the distributed platform, where producer and consumer channels are available to produce and update data for end nodes [57]. Kafka[4]® is an open-source platform; it was initially developed by LinkedIn, then became open-source (Apache Foundation).

Industrial communication networks

Industrial networks were first defined in 1981. Since then, many enhancements have been made due to the use of physical communication technology. In general, an "Industrial Communication Network" refers to a network that provides data exchange in a factory. These networks can be categorized as Fieldbus, real-time Ethernet, and wireless [58], and the data exchange can refer to machines, assets, and industrial devices. In the last few years, the need for high-speed data transmission has increased dramatically in Industry 4.0, as Industry 4.0 requires high-speed data exchange between shop floors and the cloud [58].

Industrial communication is based on well-established industrial frameworks and protocols. This section takes advantage of a set of open-source products that provide a complex yet compelling environment, using some well-known tools and emerging open-source technologies.

Profinet Profinet is an industrial network communications standard based on industrial Ethernet. It is particularly suitable for time-critical applications and provides integrated diagnosis and safety at the protocol level. The connectivity between the controller and the device is carried out by the PROFINET IO module, which represents how the data is exchanged in Profinet [59]. It is based on a consumer-producer model and can be implemented by any Ethernet controller. A high degree of availability is guaranteed, using a redundant architecture, which is essential in time-critical applications.

Modbus Modbus is an industrial protocol developed in 1979 for automation systems in industrial environments [60]. The protocol is based on a client-server paradigm, in which the client starts transmission, and the server can respond to only a particular request. Each client is directly addressable, and the server responds to only the client who began the conversation. The Modbus protocol communicates

[3]https://www.mongodb.org

[4]https://kafka.apache.org/

through various busses and networks, and it has become the de facto standard for industrial serial communication. The Modbus TCP is a slightly updated version of the Modbus protocol, accepted as a standard. It requires the TCP/IP layer to be used in network communication, either on the same network or over the Internet. This Modbus version can be used to connect to machines and PLCs.

OPC-UA *OPC* is an open standard used in the industry for secure and reliable data exchange [61]. It was released in 1996 and has become one of the most widely used platforms for exchanging data within an industrial system. It supports communication within machines, between machines, and from machines to systems. The OPC specification defines an interface between system components to permit them to share data using a unified communication standard without using a custom interface. It is cross-platform and scalable and can be used in a wide range of environments, from small embedded applications to massive cloud systems and mobile applications. The communication protocol is designed to enhance security by providing authentication and encryption and can thus be used both within private networks and across the Internet. The initial standard was released only on Microsoft systems, and it is now known as Classic OPC, while the newer standard introduced in 2008, named OPC-UA (Unified Architecture), is based on the IEC62541 standard and includes all features of the classic version, plus additional features, such as platform independence and diagnostics.

MQTT Message Queuing Telemetry Transport (MQTT) is an M2M connectivity protocol that complies with the publish/subscribe paradigm [62] developed in 1999 at IBM and Arcom. It was designed to be lightweight and straightforward and was initially intended to be deployed on low-power sensors. However, given its flexibility, it has been widely used in industrial products. Since the goal is to keep the protocol as simple as possible, MQTT does not provide specific methods to enhance security; however, it can rely on any additional security layer, like SSL. The MQTT working principle is based on topics in which nodes can publish their data as a topic on the dedicated channel. The MQTT client can subscribe to the topics to receive notifications and messages. (At the time of this proposal, MQTT is undergoing the standardization process at OASIS.)

1.2 TSCH Wireless Sensor Network: Problem Statement and Challenges

Several applications in the industry 4.0 require low-power consumption for WSN battery-powered devices, as well as low latency and high-reliability [63]–[67]. Typically, WSN nodes (also referred to as motes) are battery-powered and, in some situations, are installed in environments or locations where it is difficult or impossible to change the batteries. Thus, replacing batteries is a challenging task.

WSNs cannot be adopted for time-critical applications requiring extremely low latencies, such as a few milliseconds [68], [69]. Nevertheless, specific, high-performance wireless solutions have been recently defined for these applications: ultra-reliable and low-latency communications (URLLC) [70]. However, many contexts, such as industrial monitoring and surveillance applications, may benefit from (and even demand) the underlying network's ability to ensure short and bounded latency on transmitted packets.

Depending on the application, network reliability, such as power consumption and latency, must be guaranteed. In theory, this can be accomplished with automatic repeat requests (ARQ) [71], which rely on confirmed delivered messages and performer retransmission for frames in which no corresponding acknowledgment (ACK) is received. Power consumption and latency will be increased with the retransmission technique.

The most popular transmission technology is IEEE 802.15.4, which is adopted in WSNs [72]. It has low energy consumption, which is suitable for battery-powered devices, and it enables a flexible and straightforward implementation of a medium access control (MAC) mechanism via software.

Although IEEE 802.15.4 establishes a beacon-oriented transmission method with guaranteed time slots to improve determinism, beaconless methods provide asynchronous network access and are generally utilized in real-world applications.

A TSCH mechanism is subsequently proposed to increase the reliability of WSNs. TSCH is an enhanced MAC technique that reduces the probability that packets sent by applications are lost due to disturbance (e.g., electromagnetic noise) or interference from a different wireless network.

Depending applications demand different WSN requirements. For instance, the gas station required higher transmission reliability, and in this specific case, latency was not an issue. However, power consumption is an essential factor. Nevertheless, in other applications, such as environmental monitoring or home automation, low latency, and high reliability are not issues. Nevertheless, low power consumption is needed when the WSN node is battery-powered.

We propose a method to analyze WSNs before deployment and estimate WSN performance indicators in a specific scenario. The following chapters (Chapter 5 and 6) will pose a problem related to WSNs' communication performance indicators

and then propose a mathematical model to analyze the performance indicators. The model could also estimate the reliability and latency based on simple `PING` utility round trip experimental data. The proposed model is based on the OpenMote B+ devices, which runs the OpenWSN OS. The nodes support the 6TiSCH protocol, which is one of the latest WSN protocols. 6TiSCH is ip6 over the TSCH network, with many features to guarantee better reliability and low power consumption in the WSNs. The models in Chapter 5 and 6 are proposed based on data obtained by running the 6TiSCH protocol on the OpenMote devices. The chapter 5 focuses on the single-hop network modeling, and the chapter 6 extends the proposal to a multi-hop network. Furthermore, *TSCH predictor* is proposed for network simulation and estimates the performance indicators of simulation results of the WSN. The model is introduced to help engineers validate the WSN performance indicators before deployment. Moreover, the *TSCH predictor* is developed in a simple way to estimate and predict the WSN's behavior. The *TSCH predictor* was implemented in a user-friendly and straightforward way to perform a simulation and avoid becoming a complex simulator.

The main contribution of chapter 5 is a practical and straightforward mathematical formulation that explains a TSCH network's behavior to help tune procedures. The model depends on protocol configuration parameters and estimated quantities to characterize the network and the surrounding environment, obtained from experimental measurements performed on a real setup. The model validation was performed on the logged data from the testbed described in chapter 4.

The experimental characterization of the power consumption of OpenMote B devices was the second relevant contribution of the proposal. The motes are popular and are currently used in developing and testing newly added features or improvements of both 6TiSCH and the OpenWSN OS. However, to the best of our knowledge, such an analysis of power consumption is not available in the literature for OpenMote B devices. This characterization provides a fair and accurate estimation of the node's power consumption in the WSN.

The third contribution was implementing the TSCH predictor which estimates the real behavior of the 6TiSCH network in compliance with the actual data collected from the WSN. Experimental analysis was performed to validate the effectiveness of the TSCH predictor by comparing the data obtained from the OpenMote hardware.

1.2.1 WSN Challenges

Many devices exchange data over wireless communication; many vendors use wifi technologies to send and receive data over the air. Thus, Wi-Fi traffic is increased. However Wi-Fi devices are based on the IEEE802.11 standard, which is used in many common wireless devices as well as industrial machines to transmit data over the air [73]. These wireless devices, known as basic service sets (BSS),

are widely deployed in offices, homes, hospitals, and industrial environments.

Most BSS devices transmit data via the 2.4 GHz industrial, scientific, and medical band (ISM band). WSNs are based on the IEEE 802.15.4 standard, which uses the frequency as an ISM band. This creates a collision between the WSN and other devices working in the same frequency spectrum. One popular solution is to plan the frequencies of co-located networks to limit interference. Nevertheless, this solution is usually infeasible, as different departments manage different networks. Therefore, the WSN packets suffer from delays due to interference and collisions, and the packets may even be dropped.

Fig. 1.2 shows the impact of background traffic on the network. This experiment was performed over four days (Friday to Monday) with the 6TiSCH network set up using the `ping` utility. The 6TiSCH protocol was run on the OpenMote B devices to obtain the round-trip values. The average, minimum, and maximum statistics were computed over a sliding window. The window averaging was calculated for 120 samples, for which each ping was sent with a 30 second interval.

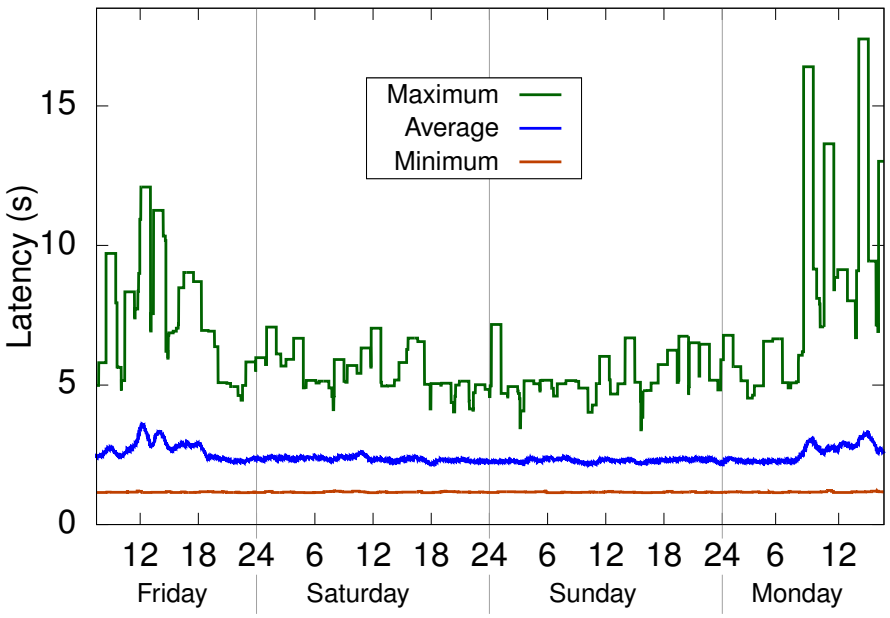


Figure 1.2: Measured `ping` round-trip time (Max./Avg./Min. values, Friday-to-Monday, 1-hour moving average).

As Fig. 1.2 shows, the latency changes on the weekend and overnight on weekdays, and it has a lower value concerning workdays. The latency increases more than three times during the workday. Furthermore, there were no high-power electrical machines presented near the testbed. This means that the communication quality was affected by background traffic. This effect was due to other Wi-Fi devices (e.g., mobile phones, PCs, access points) near the laboratory.

Fig. 1.3 shows the active BSSs, which were visible in our experimental testbed. The experimental analysis performed in this setup shows that the Wi-Fi network directly impacts the WSN (mote) communication quality. However, WSNs were not developed to support deterministic traffic, and this type of delay is acceptable for many applications (e.g., lighting systems or smart homes). Nevertheless, the TSCH mechanism provides higher reliability for WSNs, making them suitable for mid-critical scenarios (e.g., industrial plants, mission-critical applications, disaster management), where time requirements must be met [74], [75].

There are other solutions to satisfy low power and low latency, which suites the requirement of lighting-control applications in smart homes. This solution is called the wireless short-packet (WSP) protocol, and its energy consumption is optimized by using the IEC 14543-3 standard [76]. WSP solutions work well and save much energy by decreasing duty cycles, but they were not developed for the meshing network [77].

Wake-up radios are another approach in which some technology complies. In this scenario, one ultra-low-power receiver is used to wake up the primary radio by demand [78]. However, this technology is currently unavailable.

In the TSCH mechanism, the network's time is split into slotframes, in which unique nodes send or receive a packet in the scheduled slot time. In this scenario, in end-to-end communications (including two-way links), when the links are scheduled in a slotframe, they are permitted to occur within the same scheduled slotframe [79]–[81]. This means that, in theory, the round-trip time of each query based on the request-response paradigm is as low as 1–2 s. Moreover, when energy harvesting

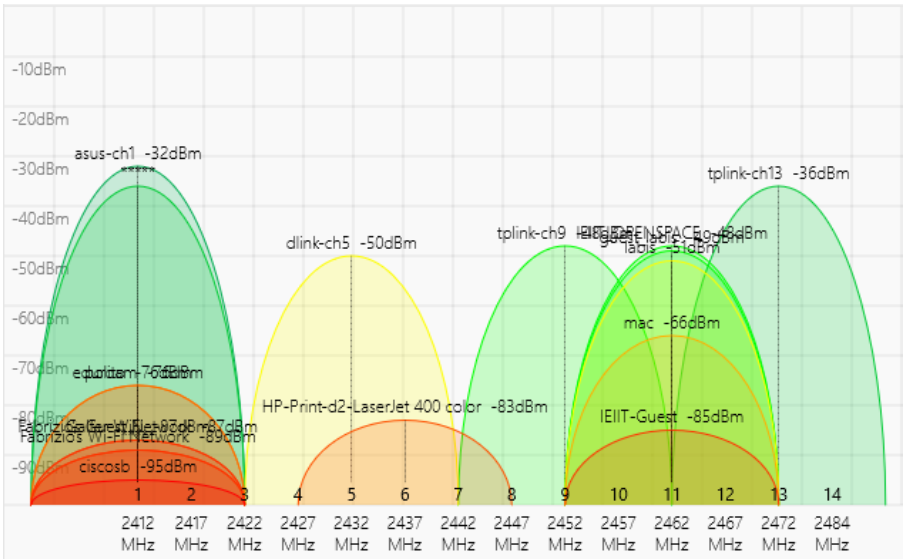


Figure 1.3: Wi-Fi spectrum traffics during experimental campaigns.

is exploited (e.g., the motes are not powered only by batteries), TSCH networks can be adopted for close control loops, with slow dynamics and high-reliability [82], [83]. Therefore, the TSCH communication network quality could be obtained to interfere with Wi-Fi traffic in real scenarios. Additionally, the number of interfering Wi-Fi is essential for deciding whether the requirements demanded by the applications can be met.

The effects of common interference between different wireless network technologies have been widely studied. Cross-interference analysis between Wi-Fi and ZigBee are studied in [84], [85]. In [86], the effect of IEEE 802.15.4 traffic on a Wi-Fi network was evaluated. The influence of Wi-Fi traffic on legacy (non-TSCH) IEEE 802.15.4 networks has been assessed in many studies, such as [87]–[90]. In addition, the Wi-Fi interference effect on 6TiSCH (based on IEEE 802.15.4 with TSCH) was analyzed in [91]. The performance of the channel hopping technique has been evaluated in [92]. Finally, the whitelist and blacklist of channel hopping have been proposed to improve the reliability in WSNs [93].

Unlike the above works, the Chapter 6 describes the mathematical model to analyze the effects of Wi-Fi interference on a TSCH single-hop network; then, in Chapter 6, the model will be extended to a multi-hop network. Most of the past literature's theoretical model considers only a single technology, excluding any interactions with other wireless protocols. Earlier investigations based on Markov chains, concerned Wi-Fi [94]–[96] which could be extended to TSCH as well. In particular, in [97], [98], analytical derivations are introduced to represent the shared cell transmission. The mutual interaction is analyzed between IEEE 802.15.4 and IEEE 802.11, which typically refer to older versions of the standards. Other papers report on theoretical studies about the impact of Wi-Fi traffic on a pre-TSCH version of IEEE 802.15.4 [99], [100]. The proposed mathematical models have been driven by the measurements obtained from a real setup in the next chapters. Therefore, the mote communication performance can be estimated by the proposed model.

It is better to know the issues in both technologies' characteristics to propose solutions for them. The IEEE 802.15.4 and IEEE 802.11 have different characteristics concerning the physical layer. The bandwidth of IEEE 802.11 is 20 MHz for a single OFDM channel (22 MHz for DSSS). However, IEEE 802.15.4 channels have 16 channels defined by the ISM band, and the frequencies are between 2.405 GHz and 2.480 GHz, with 5 MHz bandwidth for each channel.

Fig. 1.4 shows the channels and frequency overlap; every channel in the Wi-Fi link overlaps with about 4 adjacent WSN channels (or 8 when channel bonding is exploited). This means that when four Wi-Fi nodes are transmitting data over the air on fixed channels, such as 1, 5, 9, and 13, every IEEE 802.15.4 channel in the ISM band suffers from interference.

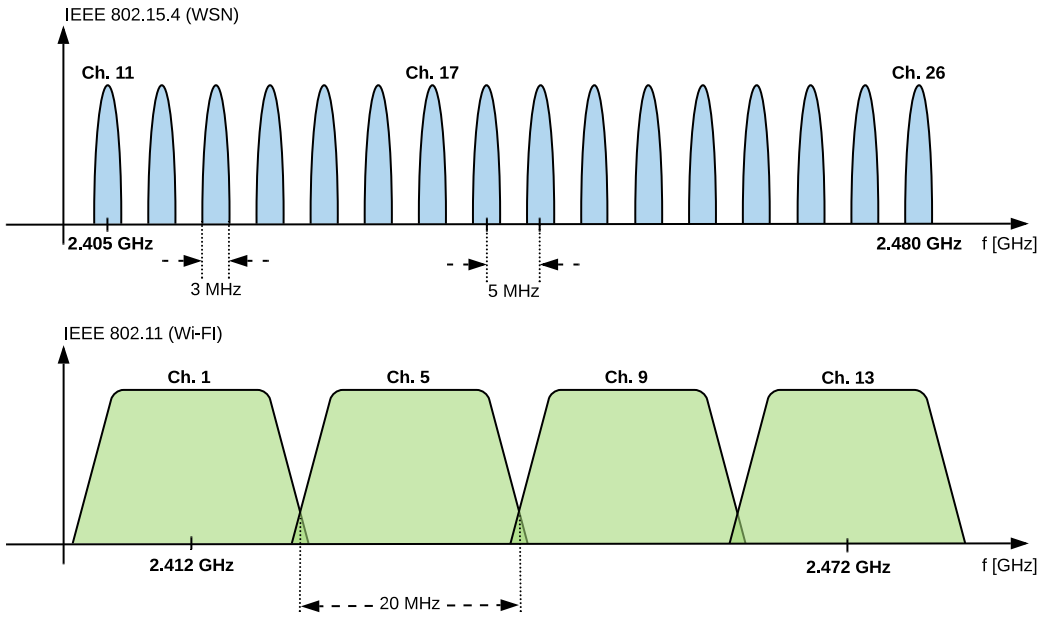


Figure 1.4: Overlapping channels in IEEE 802.11 and 802.15.4 (ISM band).

1.2.2 Related work

Many standards exist for wireless networks [101], which can be adapted to the IoT paradigm, and, often, several heterogeneous communication technologies coexist within the same system, which is transparently exploited by applications [102]. Many IoT solutions rely on IEEE 802.15.4 for frame exchanges (e.g., ZigBee and WIA-PA). Concerning IEEE 802.15.4e, two operating modes are defined: the deterministic and synchronous multi-channel extension (DSME), and TSCH (used by ISA 100.11a, WirelessHART, and 6TiSCH). Other technologies commonly adopted in IoT are LoRa, IEEE 802.11 (Wi-Fi),4G/5G, and Bluetooth Low Energy (BLE).

The proposed model in the next chapter specifically refers to TSCH, and the experimental results were obtained on motes running with the 6TiSCH protocol, the IPv6 over the TSCH mode of IEEE 802.15.4e protocol [103]–[105]. TSCH could satisfy a deterministic behavior for what concerns IoT system metrics [106]. In the next chapter, the performance analysis of the network is also evaluated. The analysis addresses, *reliability*, *power consumption*, and *latency*. Protocol behavior can be tuned through several parameters, most of which can be configured by the user. Nevertheless, since the above metrics are deeply weaved, it is generally impossible to optimize one without worsening the others. Consequently, a proper network arrangement that guarantees the application requirements must rely on heuristic approaches.

1.2.3 Time Slot Channel Hopping mechanism

The Time Slot Channel Hopping (TSCH) mechanism was first established in 2012 and was initially an improvement of the IEEE 802.15.4e [72], [107], to increase the reliability of WSNs. TSCH effectively prevents disturbances caused by electromagnetic noise or transmissions on the wireless medium presented by other communication technologies that work in the same (or on an overlapping) frequency spectrum. It also improves communication reliability and reduces energy consumption.

The TSCH network time is the key parameter that divides a network's time into slots with a fixed duration. Each slot is organized into slotframes made up of a fixed number (N_{slot}) of slots. Slotframes repeat in time and are utilized by a *time slotted* mechanism for controlling wireless medium access (MAC). This mechanism expects all motes to be time-synchronized [108], [109]. Each slot time duration ranges between 10 ms and 20 ms for common available implementations.

The following chapters explicitly consider the 6TiSCH protocol IPv6 over the TSCH mode of IEEE 802.15.4 [103]. The reasons that are used in the experimental analysis are, firstly, as mentioned before, TSCH provides noticeably higher communication reliability than legacy IEEE 802.15.4 operating modes with the same low energy consumption. In some cases, it could obtain better results in dense environments, where the intra-network implies interferences that cannot be ignored. The second reason is that IP adoption integrates sensor networks in the existing communication backbones easily and deployment faster by enabling asynchronous request-response communications via nodes.

TSCH Protocol Basics

TSCH is placed in the data-link layer and is responsible for exchanging frames between neighboring nodes. TSCH relies on two interrelated mechanisms: *time slotting* and *channel hopping*.

Time slotting

Time slotting follows a more generic method of time division multiple access (TDMA) approaches [110], it divides time into fixed-duration windows called *slotframes*, each of which includes a fixed number N_{slot} of *slots*. All slots have a fixed T_{slot}, which is selected to send a request/response/acknowledgment frame in each iteration. After the network has been configured, some slots, uniquely recognized by their location in the slotframe, are assigned to particular *links*. All links are identified by the source and destination nodes and are assigned in data exchange (the broadcast address is also provided for the destination).

Each node is synchronized with the other node of the network [109], keeps a copy of the descriptors for the links relevant to itself, and wakes up only when scheduled,

either if it has a pending packet waiting to be sent (TX action) or requires that a packet can be (RX action) received. This operation is necessary for reducing the power consumption in the network [111]. However, the main drawback of this operation is the complexity of maintaining the node being synchronized. Therefore, the challenge remains in the TSCH network.

Slotframe transmission is repeated frequently, which indicates the communications among adjacent nodes at the data-link layer, and are scheduled cyclically in a period $T_{\text{slfr}} = N_{\text{slot}} \cdot T_{\text{slot}}$. Every slot is not necessarily used in every slotframe. The 6TiSCH paradigm, IPv6 over TSCH, provides an IP layer for each node, which helps communicate through CoAP with accurate asynchronous network access. This implies that the periodicity of data transactions, as seen at the application layer, is loosely tied to the slotframe duration and structure defined at the data-link layer. This allows for easily modifying the old value at runtime without the need to reconfigure MAC parameters. Further, on-demand node access is fully supported in this mechanism [112].

Due to the slotframe scheduler, which transmits a frame in each unique slot, interference and collision between nodes are limited in the same network. A collision may happen for shared cells or cells that are not reserved for a single source node. Therefore, a random exponential method is defined to avoid a collision. Nevertheless, shared cells are not commonly used, and they are usually not applied to carry application data interchange.

Channel hopping

Channel hopping is defined to increase the reliability of the WSN. This mechanism constantly changes the transmission frequency according to a known pattern defined by the user. The WSN node (OpenMote device) is implemented with the 16 channels, which is defined by IEEE 802.15.4 in the 2.4 GHz ISM band, each of which has a width of 5 MHz, working in the range of 2.405 GHz (Channel 11) to 2.480 GHz (Channel 26).

The 6TiSCH protocol builds a matrix for each node in the network. This matrix has information related to the node in the slotframe (e.g., when the node can send data [TX-Transmite] or when it can receive data [RX-listen]). The wireless spectrum of each node is divided into frequency and time through one or more slotframe matrices. The row represents the channel number, which is 16 channel numbers in the matrices. The columns represent a dedicated slot in the slotframe [112]. Fig. 1.5 shows a TSCH matrix example, in which the rows refer to *channel offsets*, the columns identify *slot offsets*, and the slotframe utilization is defined by the TSCH matrix. Further, each *cell* in the matrix is addressed by row and column (i.e., channel offset and slotframe), represents the interconnection between the nodes, and defines when and where data should be transferred.

There are two advantages to the TSCH matrix. One prevents intra-network

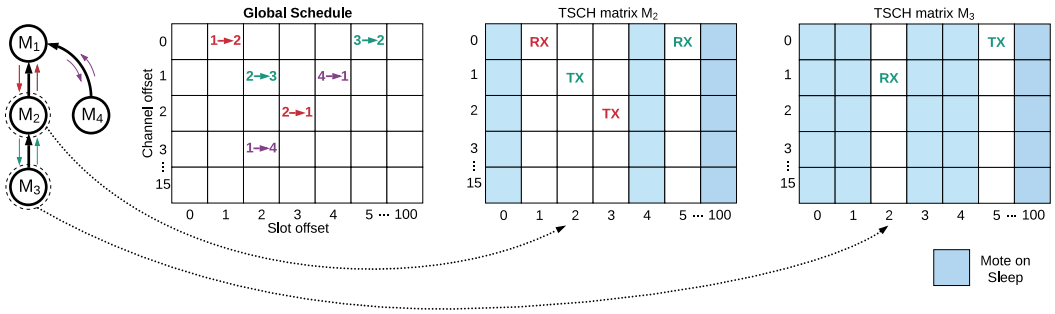


Figure 1.5: Example of a TSCH matrix defining the slotframe usage: global schedule (on the left) and local, trimmed-down copies (on the right).

collisions by configuring the desired matrix [113], [114]. The example presented in Fig. 1.6 shows a multi-hop path between $M3$ and $M1$. Each node's cell is scheduled in the matrix, with no collision between the nodes (compare the cells in the matrices $M2$ and $M3$). The second advantage is the channel-hopping mechanism, in which each cell transmits a packet on a different physical channel in each iteration. This mechanism provides higher reliability and increases the packet delivery probability in the network by avoiding collisions with other wireless networks working near a node with unpleasant traffic.

The channel-hopping mechanism is defined with a simple strategy in the 6TiSCH protocol. In this mechanism, the nodes must be synchronized; therefore, a broadcast message is scheduled to send a synchronization packet to all neighbor nodes in the 6TiSCH protocol. This means that when the nodes are synchronized, they will run with the same iteration counter number in each slotframe. Each slot in the TSCH matrix is defined by the absolute slot number value (ASN), a unique number initialized when the WSN starts. This number is unique, and it is the same for all nodes in the network. The physical channel is obtained by computing

$$PhyCh = HopSeqList[(ASN + ChannelOff)\%HopSeqLen] \tag{1.1}$$

where *HopSeqList* is the list of channels with a size of *HopSeqLen*

$$HopSeqList[] = \{5,6,12,7,15,4,14,11,8,0,1,2,13,3,9,10\} \tag{1.2}$$

and *channel offset* is the row number, and ASN is a sequential unique number in the TSCH matrix.

The physical channel is computed in each iteration to avoid the effect of background traffic generated by other networks. Fig. 1.6 shows how the physical channel is obtained; the example shows the three sequences of the physical channel in three sequential timelines. There are 16 channels available in the node (i.e., OpenWSN configuration). In the first sequence, the channel value is 0, where $ASN = 4052$, then $PhyCh = HSL[(4052 + 1)\%16] = HSL[5] = \mathbf{4}$. In the second

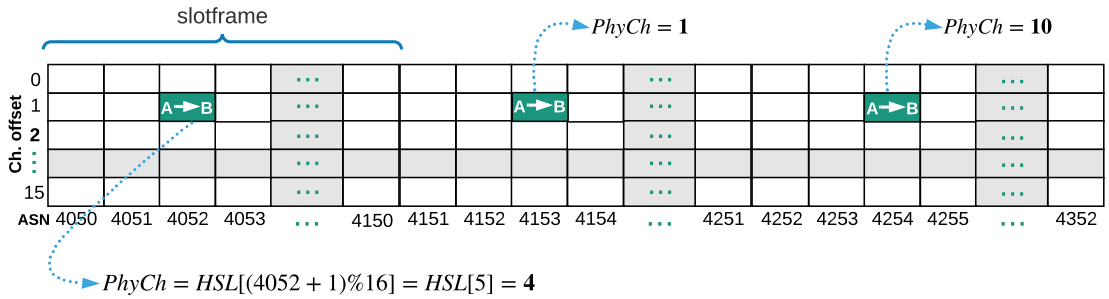


Figure 1.6: Physical channel calculation obtained from ASN counter in TSCH matrix

sequence, the channel value for the same cell becomes $ch = 1$ with $ASN = 4153$, and for $ASN = 4254$, the value becomes $ch = 10$. The counter number for ASN increases in each iteration. The ASN number is the same for all nodes in each iteration, which helps all nodes change their physical channel. This is why the broadcast message must be sent periodically based on the network configuration to keep the network synchronized.

Each cell is scheduled for transmission (TX) or reception (RX) in a single mote. When there is no activity scheduled in the cell, the mode goes into sleep mode. This mechanism saves energy in the mote where the number of slots (N_{slot}) in the slotframe is large. In the TX cell, transmission occurs when the frame is ready or a frame is waiting in the buffer; otherwise, the cell sleeps in an optimized implementation and waits for the next scheduled time. This is why the 6TiSCH network has a shallow power consumption. Further, the N_{slot} value directly affects the network latency, so it is better to select a small value for it.

The respond mechanism follows the same concept in the 6TiSCH protocol; for example, when the RX cell receives data, the mote responds after $N_{slot} * T_{slot}$ from the TX cell. This is why the 6TiSCH network needs optimization for each application. The TSCH is part of the medium access control (MAC), and the TSCH manages the communication between the neighbor nodes at the data link layer. Matrices that define slotframes and spectrum usage are configured by employing other protocols that operate at protocol stack levels.

Many published papers focus on configuring the TSCH matrix [79]–[81], [113], [115], but only a few mathematical models are proposed for describing the performance of the WSN communication. However, some theoretical analyses discussed how Wi-Fi interference could affect TSCH or DSME in the IEEE 802.15.4 networks [99], [100]. Alternatively, some papers have proposed a Markov chain model to describe the traffic on shared cells [97], [98]. Some preliminary models describing communication over dedicated cells are described in [71], but this study does not

Table 1.1: Taxonomy of the scientific works related to the WSN/WSAN.

Topic	References
WSN/WSAN applications	[12], [15], [16], [18], [20], [22], [119]–[121]
TSCH/6TiSCH	[97], [103]–[106], [112]
IEEE 802.15.4 specifications	[72], [107]
Scheduling and TSCH configuration	[79]–[81], [113]–[116]
Mathematical models	[71], [97]–[100], [122]
WSN/WSAN high-level protocols	[117] (6P), [118] (RPL), [123] (CoAP)
Power consumption	[63]–[67], [124], [125]

consider the energy consumption model and multi-hop networks. Further, effective and efficient adaptive channel selection mechanisms are proposed for channel hopping in [116], to increase the performance indicator in the WSN for eHealth applications.

The 6TiSCH protocol supports an Operation Sublayer Protocol (6P) [117], which manages multi-hop communication between a pair of nodes. This 6P permits adding or deleting cells in the TSCH network. In addition, it configures TSCH matrices and the Routing Protocol for Low-power, and Lossy Networks (RPL) [118]. Table 1.1 reports a taxonomy of the scientific works around WSN/WSAN, characterized by the related topic.

Chapter 2

Industrial IoT Platform Based on RAMI 4.0

The work described in this chapter was originally presented in [32], [126]

As discussed in Chapter 1, industrial communication protocols are proposed by industrial companies, each of which has a diverse approach to the data structure in data exchange. For instance, Siemens launched Profibus, then, by leveraging Ethernet technology, proposed Profinet. Modbus, proposed by Modicom, is a serial-based communication protocol, and it has become the de facto standard in the industrial environment. However, Industry 4.0 is an IT approach that connects machines and assets to the enterprise resource planning (ERP) system and the manufacturing execution system (MES). As always, there is a strong need for standardization, and RAMI 4.0 was proposed to target this need in the industrial community [127].

In this road map, OPC-UA (Unified Architecture) was suggested for data exchange at the factory level. The classic OPC was developed in 1996 by Microsoft for the Microsoft OS. The OPC-UA is a cross-platform protocol that enables M2M and machine-to-system data exchange by a unified standard interface. Due to its flexibility, OPC-UA is a preferred choice for many applications, ranging from embedded systems to mobile and cloud platforms. OPC-UA has an encryption mechanism, which makes it one of the most valuable protocols in the industry, as it allows for setting up private and secure networks, which is expected in any industrial protocol [128].

This chapter proposes two architectures: one is based on the Distributed Industrial Internet of things Gateway (DIIG) [32] and the Kaa[1] IoT platform, and the other is an open-source middleware that proposes an extended architecture,

[1]https://www.kaaproject.org/

including the OPC-UA protocol as the primary communication protocol. In the second architecture, industrial machines can connect directly to the proposed IoT middleware, and data can be exchanged using the OPC-IoT protocol, which is also based on the IEC 62541 standard. In this proposal, the DIIG algorithm is responsible for converting data from an industrial protocol to OPC-IoT. In addition, the NoSQL databases, such as MongoDB and Cassandra, were integrated to store data. Kafka was also adopted as a distributed streaming platform. Finally, experimental analysis was performed to evaluate throughput, round-trip delay, and fairness.

2.1 Proposed Architectures

The new IoT architecture was designed based on the RAMI 4.0 road map, and the focus of the architecture was the communication layer [25], which complies with the IEC 62541 standard. Fig. 2.3 shows the proposed architecture, which complies with RAMI 4.0. Moreover, the DIIG platform which was presented in Fig. 2.1 [32]. It was developed with the Kaa IoT middleware, which manages communication and data storage. It complies with the NoSQL database. This platform was evaluated for different databases (i.e., MongoDB, Cassandra, and Kafka, where Kafka was a streaming data platform, which enabled data streaming for the distributed platform).

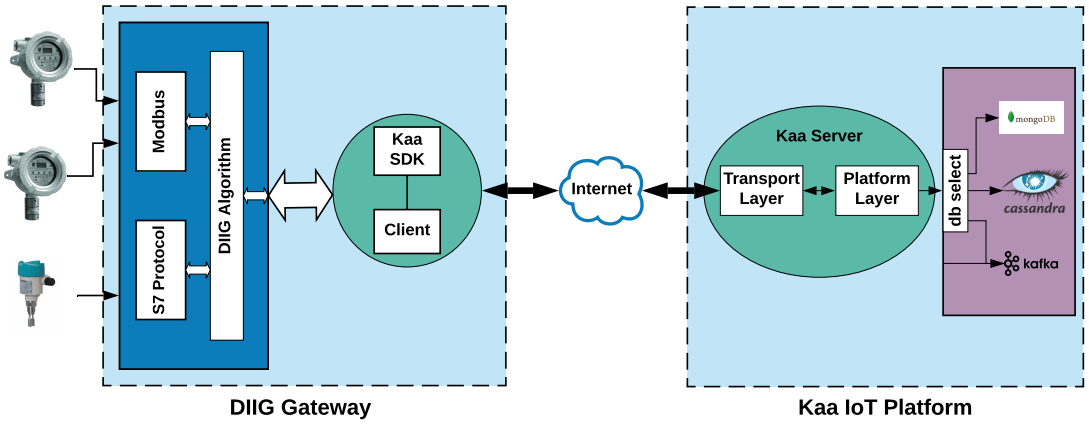


Figure 2.1: DIIG architecture components with IoT platform

2.1.1 DIIG-Kaa

The proposed platform is implemented with a DIIG gateway (more explanation about DIIG algorithm is presented in Appendix B)[32]. The platform has two main layers: a DIIG gateway and Kaa IoT middleware. The DIIG gateway transmits data from the shop floor to the Kaa IoT platform. Fig. 2.1 shows the data exchange

between the DIIG algorithm and the Kaa client routine. In this routine, when data are ready in the gateway, the client program sends the data to the server by utilizing Kaa SDK in the gateway. The server is set up by open-source industrial Kaa IoT middleware, which collects data and stores them in the NoSQL database. In this chapter, the performance evaluation was obtained by MongoDB, Cassandra, and Kafka data storage.

Fig. 2.1 shows that the DIIG gateway can communicate with nodes supported by the Modbus and S7 protocol. This section aims to discover the best database among the available technologies (i.e., a technology that can store data at low latency and higher throughput using the DIIG protocol). The DIIG algorithm has the advantage of providing a communication channel with the industrial network, such as Profinet and Modbus protocols. Furthermore, the DIIG protocol can exchange data between nodes and the Kaa IoT middleware without considering the nodes' data generation protocol (Modbus or the S7 protocol).

As Fig. 2.2 shows, $DB_m[256]$[2] byte was allocated for each client session (IoT Node), which was partitioned into an eight-byte memory block per message. The ***Tr*** and ***Ts*** processes were implemented to control the write/read mechanism in the memory blocks.

The ***Tr*** process checks the memory status flag (MSF) block when the block is free, and the ***Tr*** process writes the memory address in the free memory index (FMI) block. If the memory block is full, the ***Tr*** process writes the memory index in the ready-to-send index (RTSI) block. Both processes run in parallel. The client node also must identify the memory index in the MSF when it writes new data to the memory MSF block.

The ***Ts*** process was implemented to send data to the Kaa IoT platform, and it checks data by the RTSI flag. Then, if the data are ready, the ***Ts*** process sends them to the Kaa client.

The Kaa client was implemented by the Kaa endpoint SDK, which was generated by the Kaa platform. The Kaa platform is designed for scalable and flexible applications. This advantage provides a platform that can customize the data sources and data storage system by configuring the project. It is also possible to produce a unique development SDK for each application. The Kaa server provides the data marshaling, communication, and notification control within the Kaa APIs. The client application in this gateway is based on the Kaa SDK; after all, the client was developed to read data from the DIIG gateway, and it transmits the data from the DIIG algorithm to the Kaa platform. Finally, the data is stored in the target database using the Kaa IoT middleware.

In this chapter, a different database is selected to store data. As shown in Fig. 2.1, MongoDB and Cassandra are selected as NoSQL DB, and Kafka is chosen

[2]Data Block Memory

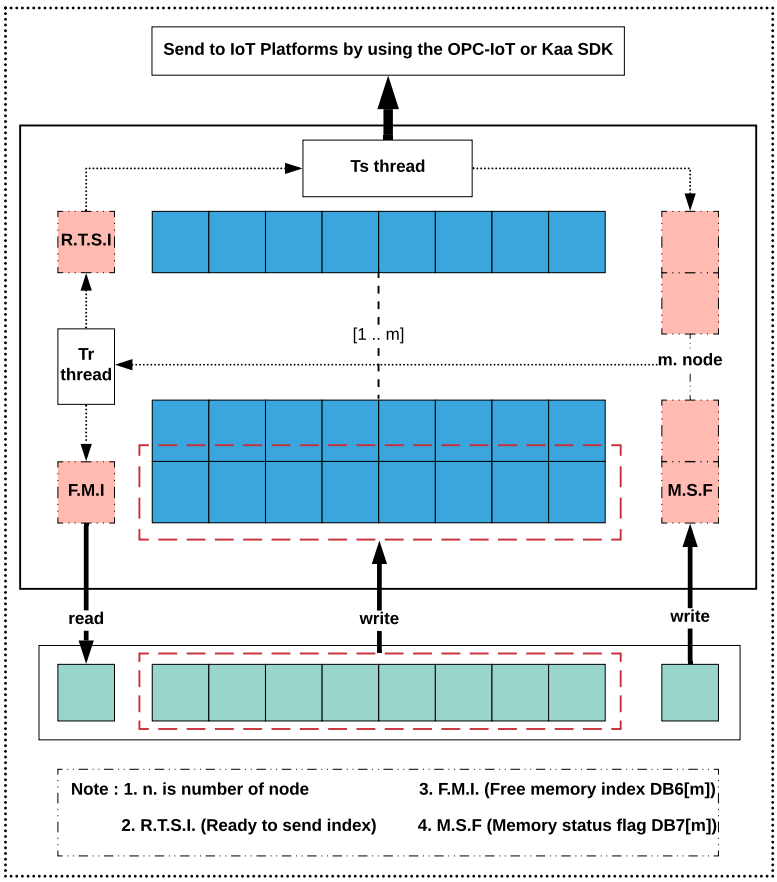


Figure 2.2: DIIG gateway algorithm

as a distributed streaming platform for broadcasting information between several applications. The design choices are audible to each other, and the data structures are NoSQL.

2.1.2 OPC-IoT

The OPC-IoT is proposed based on the OPC-UA protocol for the communication core, which complies with the IEEE 62541 standard, and it is a service-oriented architecture (SOA).

As shown in Fig. 2.3, the two stages are implemented (i.e., the *DIIG-OPC*[3] gateway and *OPC-IoT middleware* platforms). The OPC-UA SDK is employed

[3]DIIG: Distributed Industrial Internet of Things gateway

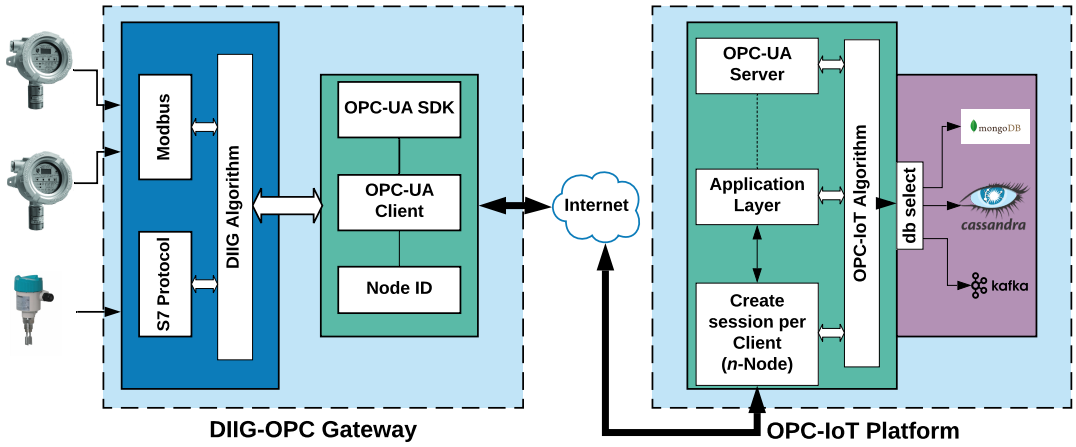


Figure 2.3: DIIG-OPC architecture components with the IoT platform

to develop the DIIG-OPC gateway instead of the Kaa SDK, based on the DIIG-Kaa architecture. The DIIG algorithm is implemented to communicate efficiently between the gateway and industrial assets. Further, in this gateway, the client is developed by the OPC-UA protocol. As shown in Fig. 2.3, two primary core areas in the OPC-IoT middleware are *communication* and *data storage*. Four layers are implemented in the communication core: the application, the OPC-UA server, the OPC-IoT algorithm, and the client session controller.

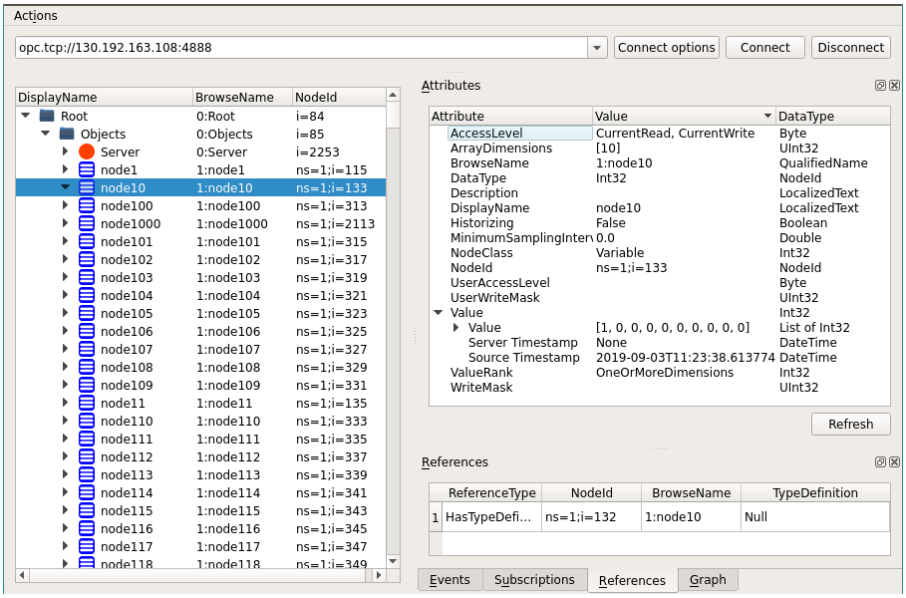


Figure 2.4: Server & Data object in the OPC-IoT platform (for 1000 clients)

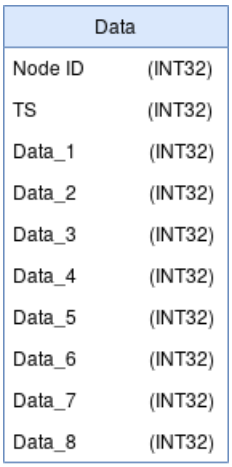


Figure 2.5: DIIG-OPC database schema for the IoT platform.

The OPC-UA protocol is employed differently in this architecture; the OPC-UA server was developed to collect data from clients, which is uncommon in this protocol. The server is implemented to collect data from clients, and each client creates a connection with the server to send data to the dedicated node specified by the OPC-UA server. Then, the middleware stores the data in the database(s). Fig. 2.4 shows the server structure; each client has a dedicated node on the server-side.

The round-robin algorithm manages data in the OPC-IoT platform as "first come, first served," which means no priority is defined for serving data in the node management. After data marshaling, the data is stored in a NoSQL database. In this proposal, the platform complies with MongoDB and Cassandra. Furthermore, it uses Kafka as distributed data streaming, which provides a real-time channel to broadcast data between different parties. The complimentary driver is implemented for all three mentioned data management systems. Fig. 2.5 shows a general data schema stored in the database.

The DIIG algorithm provides a link between the industrial network protocol and the gateway. When the data is ready, the T_s process writes data on the OPC-UA server. As shown in Fig. 2.3, four stages were implemented to send and store data on the server: (1) read instrument values, (2) send data to the IoT middleware, (3) process data on the server, and (4) store data in the database(s).

Read instrument values

The DIIG algorithm was employed to read data from the industrial network, such as Profinet and Modbus protocols, on the shop floor. The gateway provides Profinet master, and it dedicates 2048 bytes of memory data blocks for each client. Thus, a PLC or other industrial IO could transfer their data to the gateway memory block directly. As explained previously, the DIIG algorithm reads the available

data, changes the data format, and transfers them to the IoT protocol schema, then sends the new data (in a new form) to the IoT middleware.

Send data to the IoT middleware

The data is sent to the IoT middleware by the OPC-UA protocol. The DIIG algorithm receives the data, and then it sends it to the dedicated node on the OPC-IoT server. Each client has a unique Node ID that the OPC-UA server allocates. We implemented the server by the OPC-UA[4] library, with some modification in the library to utilize for our purpose.

Process data on the server

In this stage, the OPC-UA server accepts the incoming connection and provides a session for each client. The data is then written to the dedicated object on the OPC-UA server, using the OPC-UA client. There is a flag for each node that shows if the data block on the server is ready to write, and the client will write to the node when the flag is set to 1; however, if the flag is set to 0, the data block will be blocked, and the client will not be able to write the new value to the servers' node until the status changes.

Store data in the database

The OPC-IoT algorithm was implemented to change the incoming data format. This routine sends data to the database when the data is ready in the OPC-UA server's node. In this step, the data format changes to the destination database format(s), then stored in the database. After the data are stored in the database, the algorithm modifies the status to 1, which means the server is ready to serve the next incoming data. It is worth mentioning that there is a poll mechanism in the database driver, and it is fast enough to serve the incoming data from the OPC-UA server.

Fig. 2.6 shows the two algorithms, the DIIG-OPC and OPC-IoT, developed in this work. These algorithms exchange data from the shop floor to cloud storage(s) (database[s]). The algorithm supports the delivery mechanism, meaning a message is transferred to the shop floor when the data are delivered successfully in the database(s).

[4]www.open62541.org

2.1.3 DIIG-OPC algorithm

The DIIG-OPC algorithm is presented in Fig. 2.6, which is extended from the DIIG algorithm presented in [32]. The main contribution of DIIG-OPC is that each reader (in the gateway) sends data in parallel to the server, and they are not blocked in the client process. Initially, the instrument starts communicating and reads $DB6[m]$, which stores free memory indexes. Then, the instruments write their data to $DB_m[32 * (Index - 1)]$ ... $DB_m[(32 * (Index - 1)) + 8]$, where m represents the instrument number. Each instrument can write eight logical values in memory. After writing to the gateway's memory, the memory index is stored in $DB7[m]$. These two memories are shared between the OPC-UA client and the DIIG gateway.

In the next step, the OPC client connects to the dedicated node ID, representing the instrument object (thanks to the supporting object in OPC-UA) on the OPC-UA server. Each node starts on the OPC-UA server when the server is initialized. In the gateway, the OPC-UA client checks DB7[m] (full index value) in a subroutine. It then calculates the memory address based on the index value, $32 * (Index - 1)$. The OPC-UA client reads eight values from the DIIG, after which the client checks the node m's flag on the server and, if the flag is equal to 1, sends data to the server using the OPC-UA SDK. The OPC-UA client sends a delivery message to the DIIG process, after which the DIIG process changes the status of the DB6[m] memory address to a free index address. Then the DIIG process starts over.

The OPC-UA client process continuously checks the free memory address of node m and, when it is free, transmits the value of $DB_m[32 * (Index - 1)]$ to node m to the server. The value is sent to the node m if and only if the last transmitted value is received in the server-side data storage. This system was designed to deliver data to cloud storage and notify the client.

2.1.4 OPC-IoT algorithm

The OPC-IoT algorithm is presented in Fig. 2.7 and designed to manage and store data on databases. The server complies with the IEC 62541 standard, and it establishes the node IDs on-demand on the server-side, which means it can generate the node on the fly as a request. When a connection is established between the gateway and the server, the algorithm assigns the dedicated node ID m to each instrument. The flag is available for the status of each client. The flag status allows the node to accept a new value from the client. Initially, this flag is set to 1, which means the node can accept the new value. The gateway reads values at the scheduled time, and when the flag changes to 1, the client transmits the new value to node m on the server.

Furthermore, the process checks this flag, and when the flag status changes, the process reads all node values at once, changes the original data format, and then

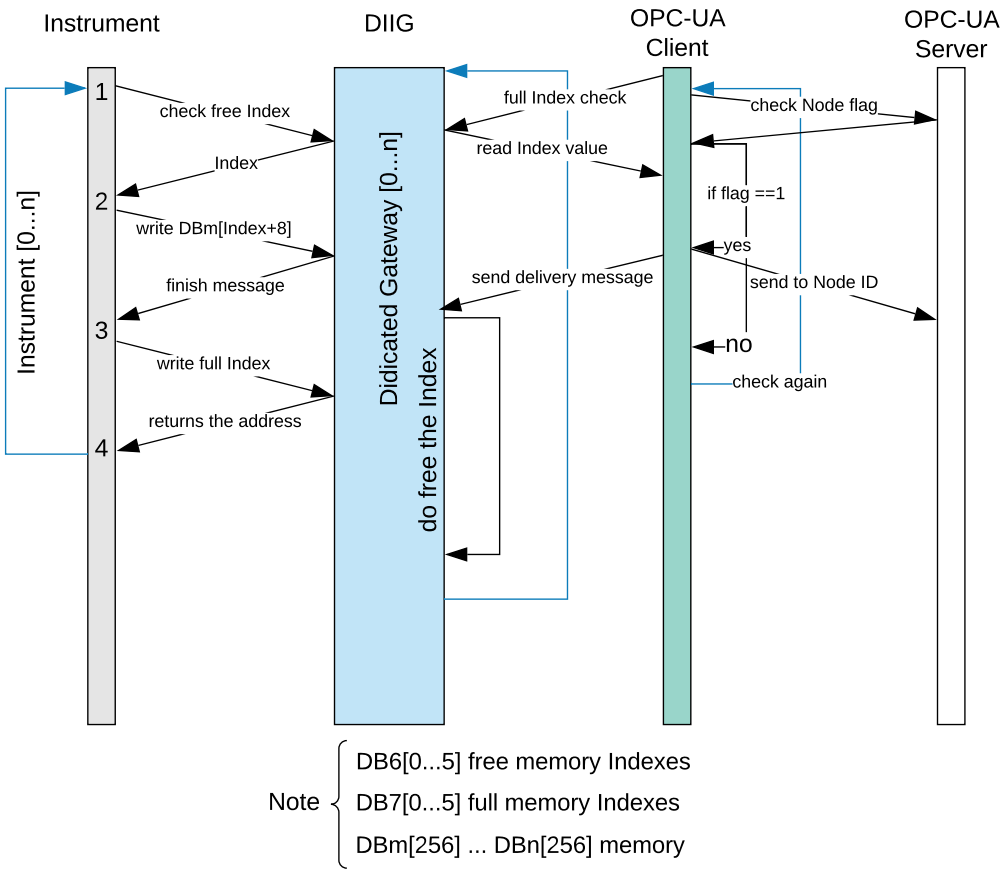


Figure 2.6: DIIG-OPC gateway algorithm

stores node m's data to the database(s). This process runs for all nodes in parallel. The delivery message is sent when the data are stored in the database(s). When the OPC-IoT flag is 1, the data has been stored in the database, and it waits for new data from the gateway(s) and client(s).

Fig. 2.3 shows that different data storage technologies are adopted, such as MongoDB, Cassandra, and Kafka. It is possible to select one, two, or three data-storage systems in parallel, and there are synchronous functions in all methods. Synchronized functions are often called blocking functions, and they are implemented to avoid the *mutex* problem between shared memories; for this reason, synchronized methods often are called in this implementation, the memory is unblocked when the task has been completed.

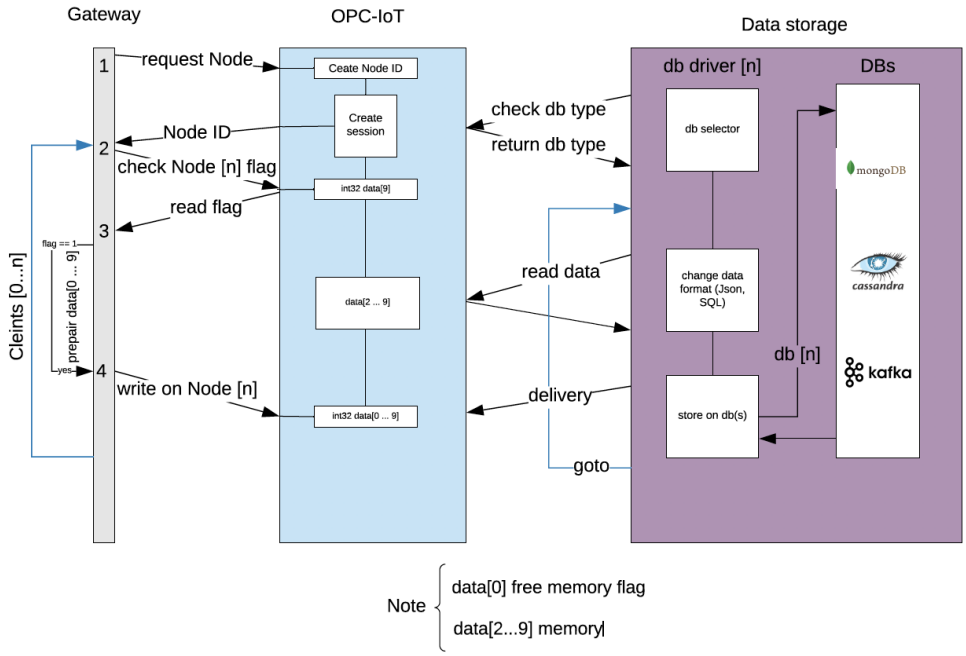


Figure 2.7: OPC-IoT Server algorithm

2.2 Performance Evaluation and Comparison

The experimental results were performed on the testbed presented in Figure 2.8. The hardware hosting server and clients must provide enough resources, so a multicore CPU was selected to host the server. The server had an Intel® Core™ Xeon(R) CPU E3-1245 v5 3.50 GHz x 8 cores, with native parallel capability. Two virtual machines (VMs) were employed for the gateways on the other PC to simulate gateways and clients. A PC with Intel® Core™ i7-7700 CPU @ 3.10 GHz x 8 cores was selected to host both simulated environments. Both were simulated and run inside the VMs on the same host. The data is produced in the profinet protocol by each client. The VMs were linked with a D-Link 1 Gigabit Ethernet switch. Fig. 2.8 shows the hardware setup. Nevertheless, four clients sent data in parallel to ensure that the server was under stress.

Parameters and setup configurations

The throughput was obtained by evaluating the total number of executed operations in each unit of time. This was the scalability indicator for the design architecture. Fairness was obtained to compare how much the proposed algorithms were fair to serve to all clients. The round-trip time was also analyzed for a time-critical scenario. As explained in the previous section, in this architecture, all

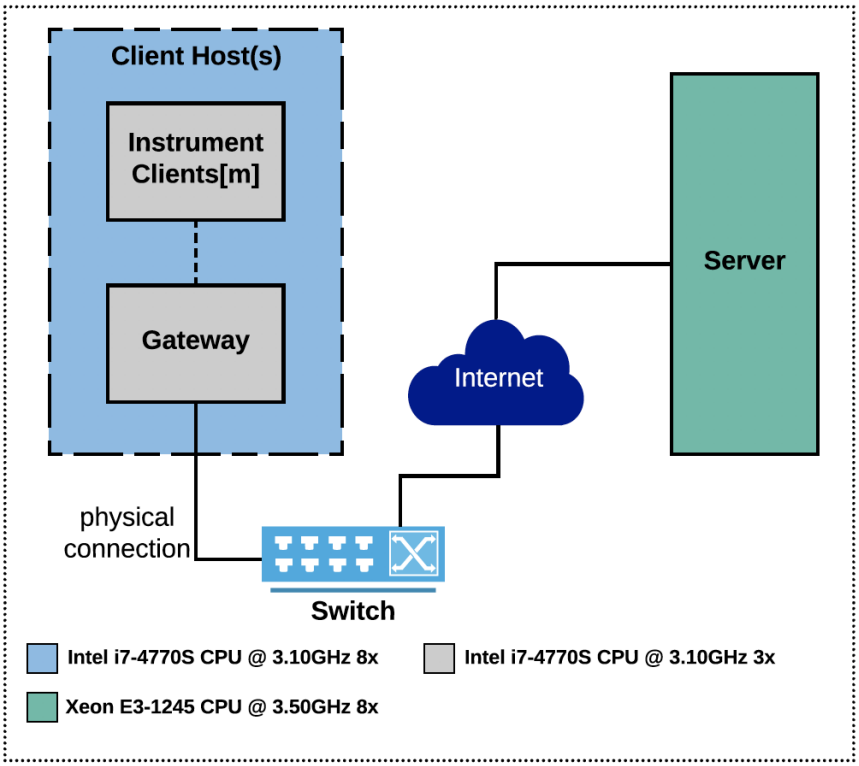


Figure 2.8: Test-bed configuration

packets include the acknowledge mechanism for all transmitted data; hence, it is possible to calculate the packet delivery time for all transmitted packets.

As shown in Fig. 2.8, the client's instrument was hosted on the VMs, which could emulate four instruments for each client. The clients could connect to the gateway hosted in the identical VMs. The experimental evaluation was performed using Kaa-DIIG and OPC-IoT middleware. The same instrument emulator was utilized for both experiments.

The throughput was analyzed for both systems. We sent 8 bytes 20,000 times from the instrument to perform this analysis. Then, the throughput was obtained by calculating the $N_{throughput} = \frac{N_t}{T_t}$, where N_t was the total data size and T_t was the total time in seconds. The experiment was repeated 5 times, and the total amount of data transmitted was 100,000. Moreover, the round-trip analysis was evaluated to examine the total time required for each message to travel from the client to the gateway and then to the server. A flag was activated when all messages had been received on the server-side.

The round-trip was calculated by dividing the total travel time for all messages over the total number of messages ($T_{roundtrip} = \frac{T_t}{N_t}$) that were sent to the database.

The minimum and maximum round-trip times were evaluated, and each experiment was performed for 1,000 messages.

Fairness is a server indicator that shows how much the server algorithm is fair among the clients. Thus, this experiment is essential for determining the effectiveness of the system. The fairness analysis was evaluated on the same testbed, and all experiments were repeated five times to reduce measurement error.

2.3 Results

Fig. 2.9 shows throughput and round-trips by the total number of four clients. Each client transmitted 100,000 messages to a gateway, after which the gateway changed the format and transmitted the data to the IoT middleware. Experimental analysis was performed using the OPC-IoT and Kaa middleware. As shown in Fig. 2.9, Kafka, Cassandra, and MongoDB were adopted in mixed scenarios.

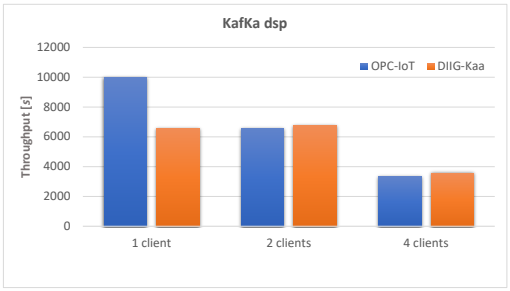
2.3.1 Throughput

As shown in Fig. 2.9, when one client connected to both IoT platforms, the throughput value varied between 9,000 and 10,000 per second for IoT-DIIG, and it stayed between 5,000 and 7,000 per second for DIIG-Kaa. In the OPC-IoT, when the number of clients increased to four, the throughput decreased by 20% to 60% in the worst cases. In the DIIG-Kaa, the throughput decreased by 35% to 45% in the worst cases. Fig. 2.9c shows that the best throughput occurs for the Cassandra database, which is a better choice in both architectures, and it also has a higher throughput on the OPC-IoT architecture platform. However, as shown in Fig. 2.9, the OPC-IoT performs better concerning DIIG-Kaa.

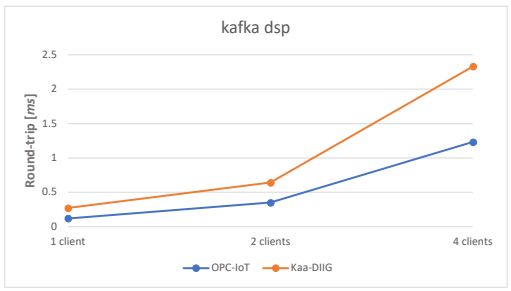
2.3.2 Round-trip

The round-trip values for both systems are presented in Fig. 2.9. The values stayed between 0.119 ms and 1.23 ms by utilizing the OPC-IoT middleware and from 0.129 ms to 1.75 ms with the DIIG-Kaa platform. There was a significant difference between the two architectures when more than one client was connected to both. As expected and presented in Fig. 2.9c, when one client was connected, the round-trip value was 0.119 ms per message on the Cassandra database on the OPC-IoT platform, and, with four clients, the round-trip value went up to 0.578 ms for the same scenario and configuration.

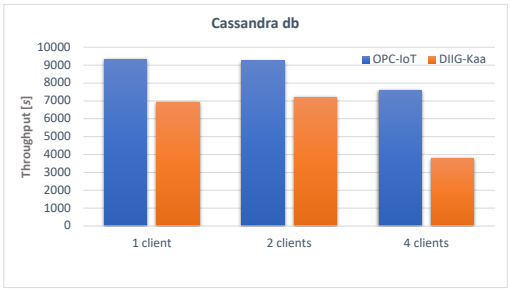
Fig 2.9 shows that the DIIG-IoT's round-trip result was significantly higher than that of the OPC-IoT. This difference was due to the implementation of a protocol that had less overhead in the OPC-UA implementation. Moreover, the storing layer of the proposed algorithm in the OPC-IoT was implemented in parallel on



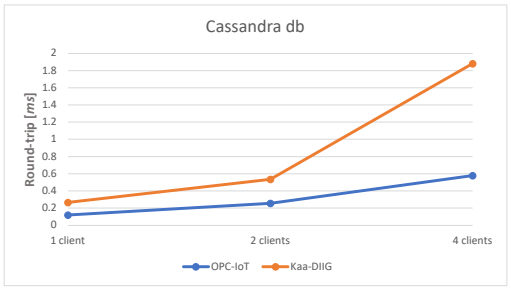
(a) KafKa® throughput



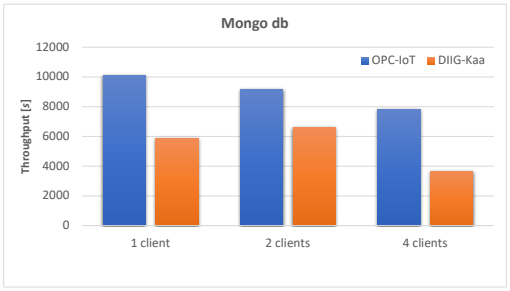
(b) KafKa® round-trip, ms



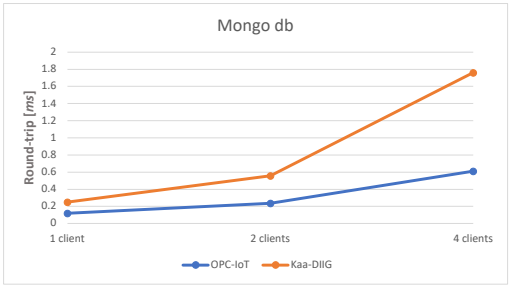
(c) Cassandra throughput



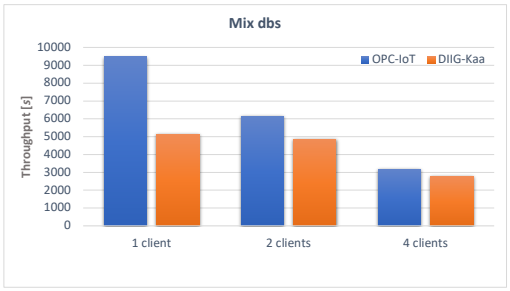
(d) Cassandra round-trip, ms



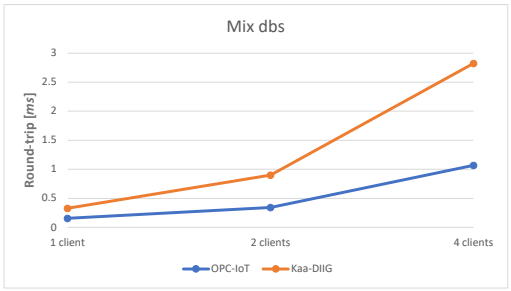
(e) MongoDB throughput



(f) MongoDB round-trip, ms



(g) Mix databases throughput



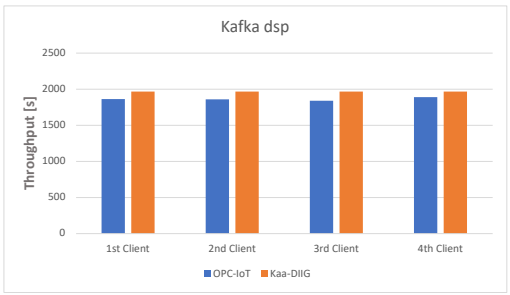
(h) Mix databases round-trip, ms

Figure 2.9: Result of 100,000 data that were sent to the server from each client, and the data were stored in various database technologies

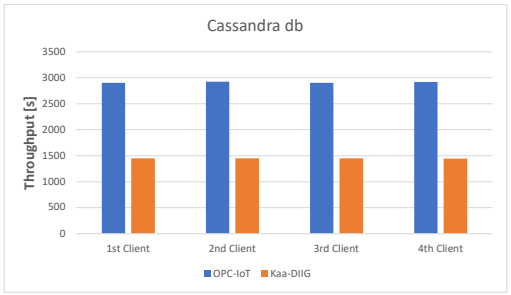
the server-side, and this was an advantage in the OPC-IoT beside the Kaa project platform, which follows a serialized mechanism.

Finally, as presented in Fig. 2.9e, MongoDB had a lower round-trip value in the OPC-IoT when four clients were sending data concurrently. Moreover, the Cassandra database had essentially the same round-trip value as the MongoDB. Thus, the round-trip value in DIIG-Kaa was considerably more than that in OPC-IoT when there were four concurrent clients. The minimum and maximum round-trip values are reported in Tables 2.1, 2.2, 2.3 and 2.4.

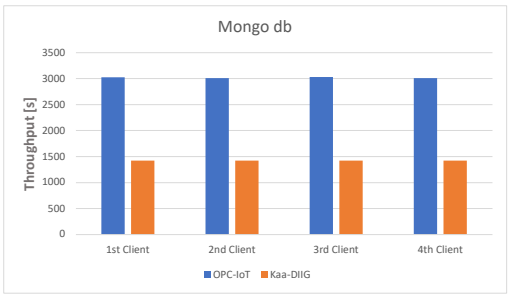
Considering the round-trip analysis, both Cassandra and MongoDB were efficiently adopted for the OPC-IoT platform and the DIIG-Kaa platform. Additionally, there was an advantage when OPC-IoT middleware was used due to the lower value of round trips and higher throughput than the Kaa middleware.



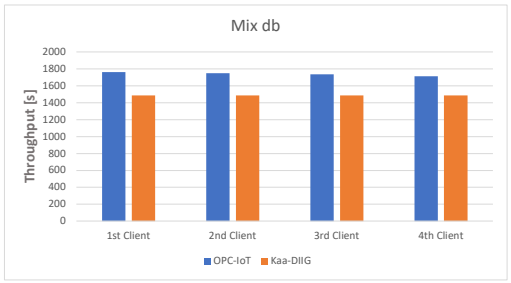
(a) KafKa® fairness



(b) Cassandra fairness



(c) MongoDB fairness



(d) Mix databases fairness

Figure 2.10: Fairness results

2.3.3 Fairness

Fairness guarantees traffic regulation in the network and server overload. To perform the analysis of fairness, we used an application to stop the experiment after 10 seconds. We then evaluated the total number of messages that were received in the database. This experiment was performed for all scenarios. Fig. 2.10 shows

KafKa	OPC-IoT			DIIG-KAA		
	min $[ms]$	avg $[ms]$	max $[ms]$	min $[ms]$	avg $[ms]$	max $[ms]$
One client	0.099	0.120	45.448	N/A	0.151	100
Two client	0.116	0.351	48.880	N/A	0.291	292
Four client	0.375	1.231	128.007	N/A	1.098	182

Table 2.1: Round-trip test in $[ms]$ for KafKa with minimum, maximum, and average values

MongoDB	OPC-IoT			DIIG-KAA		
	min $[ms]$	avg $[ms]$	max $[ms]$	min $[ms]$	avg $[ms]$	max $[ms]$
One client	0.124	0.120	35.771	N/A	0.129	7116
Two client	0.102	0.236	31.178	N/A	0.320	5075
Four client	0.084	0.609	23.104	N/A	1.151	13650

Table 2.2: Round-trip test in $[ms]$ for MongoDB with minimum, maximum, and average values

that, in all scenarios, the system was approximately unified. This behavior was due to implementing a dedicated thread for each client in the server and on the gateway side.

2.3.4 Scalability

Scalability analysis was performed on both systems. Scalability analysis provides a crucial point for choosing the IoT platform and analyzes the large size of the network in the IoT applications. For this reason, the throughput analysis was performed for 10–1,000 clients using the OPC-IoT middleware. The client could accept 3,000 clients concurrently by employing the same hardware configuration on the server-side, which was enough for an industrial IoT platform in the Industry 4.0 domain. Furthermore, by increasing the hardware resources, it was also possible to increase the number of clients. Fig. 2.11 shows that the OPC-IoT throughput was 7,300 packets/sec for 1,000 clients when the Cassandra database was employed. The experimental analysis demonstrated that, after increasing the number of clients, the system could guarantee a high number of concurrent clients on the same hardware configuration. However, the resources were inadequate for a large network, for which it is essential to use sufficient resources.

2.4 Conclusion

The work presented in this chapter was developed for two purposes: to evaluate the best database to use with the Kaa project platform and to propose and evaluate OPC-IoT on the same testbed. The results showed that the Cassandra database

Cassandra db	OPC-IoT			DIIG-KAA		
	min $[ms]$	avg $[ms]$	max $[ms]$	min $[ms]$	avg $[ms]$	max $[ms]$
One client	0.133	0.119	35.771	N/A	0.145	247
Two client	0.104	0.255	33.592	N/A	0.278	59
Four client	0.069	0.579	22.840	N/A	1.300	167

Table 2.3: Round-trip test in $[ms]$ for Cassandra db with minimum, maximum, and average values

Mix dbs	OPC-IoT			DIIG-KAA		
	min $[ms]$	avg $[ms]$	max $[ms]$	min $[ms]$	avg $[ms]$	max $[ms]$
One client	0.085	0.158	34.966	N/A	0.172	1213
Two client	0.0875	0.345	58.382	N/A	0.553	36689
Four client	0.068	1.069	24.540	N/A	1.750	56183

Table 2.4: Round-trip test in $[ms]$ for Mixed db with minimum, maximum, and average values

and MongoDB had the same performance concerning throughput; however, the Cassandra database was better when the total number of concurrent clients was increased. The test was performed using the Cassandra database for 1,000 clients, and the results showed that the total throughput was around 8,000 packets / second after increasing the number of clients. This IoT architecture was developed based on the OPC-UA protocol, which was suggested by RAMI 4.0 for a communication layer, and it enables communication with shops without any additional gateways or devices. Moreover, the gateway was proposed for data conversion using a different protocol in this work. As a result, it is possible to communicate with industrial protocols, such as Modbus or Profinet. The performance evaluation was performed with Profinet, and the delivery acknowledgment is an advantage for this architecture.

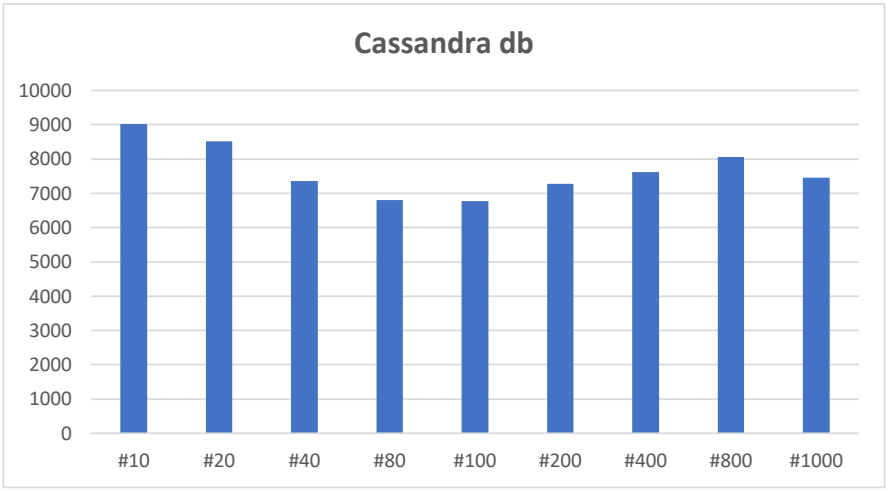


Figure 2.11: OPC-IoT: Scalability test for 10–1,000 clients

Chapter 3

Industrial Fog Architecture Based on Industrial Protocols

The work described in this chapter was originally presented in [129]

As discussed in Chapter 1, one of the most used models in IoT technology is to deploy sensitive tasks on nodes near the end-user application. This model, best known as *fog computing* [37], leaves a centralized cloud computing concept heading in a decentralized architecture, where edge nodes perform essential tasks. This paradigm decreases latency in time-critical applications, such as controlling a robotic arm or adaptive manufacturing, leaving regular computing to the central cloud services to have faster and reliable computing [38].

The decentralization of computing produces new problems surrounding cloud or fog services deployments. First, security is adversely affected by the unloading of [39] edge-node computation. In comparison, it becomes more difficult to run multiple services across the network without a quick and efficient way to perform computing in distributed modules [40], [41]. Further, most available solutions are currently proprietary and require subscriptions to be fully used.

Recently, several fog architectures have been suggested in the literature, based on the Perdue Reference Model (PRM) [130], [131]. However, the IFog4.0 architecture proposed in this chapter is built on the RAMI 4.0 standard as well as the PRM model. The proposed architecture is developed to employ the information and communication layer in RAMI 4.0. It offers the edge-computing infrastructure by including the Docker virtualization framework. In general, this architecture is designed to create an edge-computing platform for small and medium enterprises (SMEs) with high security for industrial needs. In addition, PLC connectivity is provided in this architecture. This architecture illustrates how open-source resources can help SMEs comply with Industry 4.0 requirements, and it also helps deploy quick and straightforward edge-computing components on shop floors.

3.1 Background

3.1.1 Docker virtualization

The Docker is a virtualization software that shapes the platform as a service (PaaS). The Docker utilizes virtualization at the OS-level and delivers software and new services in packages named containers. Containers are detached from each other, and work independently. Services are running on top of the OS, and act as a single virtual machine, interacting through the network connections. The Docker platform assigns local IP to each container to be able to communicate with other services or containers.

Furthermore, the Docker is able to arrange a package as if it was an application together with its dependencies all grouped into a virtual container that can run on any Linux, Windows, or macOS computer. This helps use Docker as a platform to run different software on the same operating system without installing dependencies. Hence, this proposal is based on the Docker platform, and utilizes docker as a software package management.

3.2 Proposed IFog4.0 Architecture

The proposed fog architecture, ***IFog4.0***, is presented in Fig. 3.1. This architecture is implemented by the Linux kernel v. 4.15 and is developed to comply with industrial specifications. As shown in Fig. 3.1, the ***IFog4.0*** architecture is developed with the following components: *Fog-Management*, *Docker virtualization*, *IDE*, *visualization*, *Enterprise resource planning* (*ERP*), and *industrial communication drivers*.

3.2.1 Architecture

IFog4.0 is an industrial fog platform based on RAMI4.0 standard [48]. It has been designed using a modular approach that allows to easily install and deploy new components and tools. This way, it is possible to add new features, depending on the end user needs.

The architecture is divided into four layers: Operating system, virtualization, Fog-Management system and Network Layer. Each component has been carefully selected among a set of available choices, considering three major features: flexibility, modularity and availability of open source license. Moreover, for critical sections - like the Industrial Communication Driver - throughput, latency and fairness constraint were taken into account. The operating system is based on Linux server and it features a custom Industrial Communication Driver kernel module, which has been developed specifically for this architecture. The high flexibility

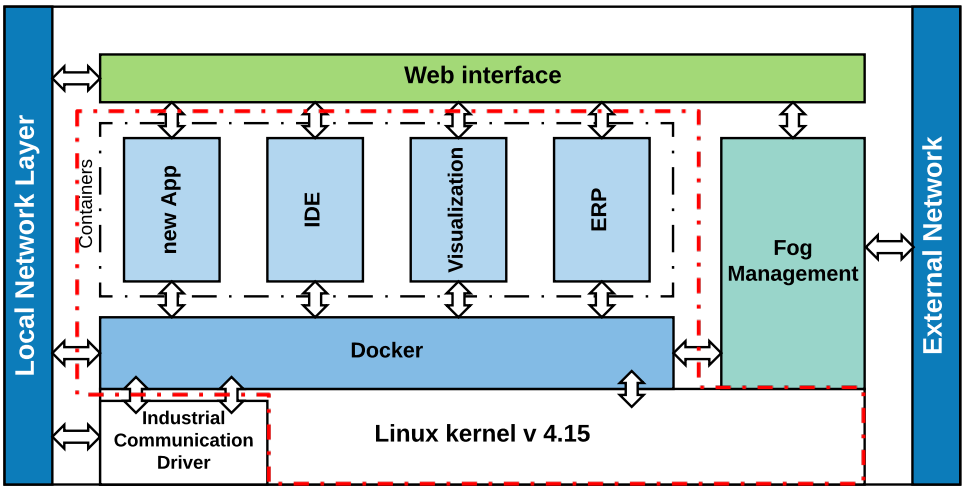


Figure 3.1: IFog4.0: Industry 4.0 open source fog architecture

provided by the Linux kernel environment was the main reason for choosing it, as it easily allow to integrate custom kernel modules.

The Fog-Management system (the green box in Fig. 3.1) is running on top of the operating system and lays directly over the kernel on OS. The virtualization layer is based on Docker platform (highlighted in blue in Fig. 3.1), and it utilizes the virtualization method to provide additional functionality to IFog4.0 architecture. Docker container system is widely used in the production environment and it's open source. There are other solutions based on either proprietary software or on less stable architectures - like Linux Containers (LXC) - which are not suitable as they would require further customization steps to improve stability. Docker allows developers to easily design and deploy new components (light blue boxes) to the architecture. The Network layer is separated from other layers, as the industrial physical layer may be based on different standards like *serial communication*, *Hart* or *CAN bus*. The modules within the red dashed box are open source tools.

As Fig. 3.2 shows, there are two different types of modules that can be developed and deployed to the IFog4.0 platform: a *component* is an entity which is developed offline using a dedicated SDK and that runs as a Docker container right above the virtualization layer, whereas an *application* is a module instantiated and executed within a component. Applications are developed directly on the IFog4.0 platform, using the tools provided by the underlying component.

In order to present the IFog4.0 architecture, some main components need to be installed, like the Fog-Management module, the programming tools, data visualization, ERP (Enterprise Resource Planning), Data Storage and Industrial Communication drivers. These modules are provided by IFog4.0 repositories.

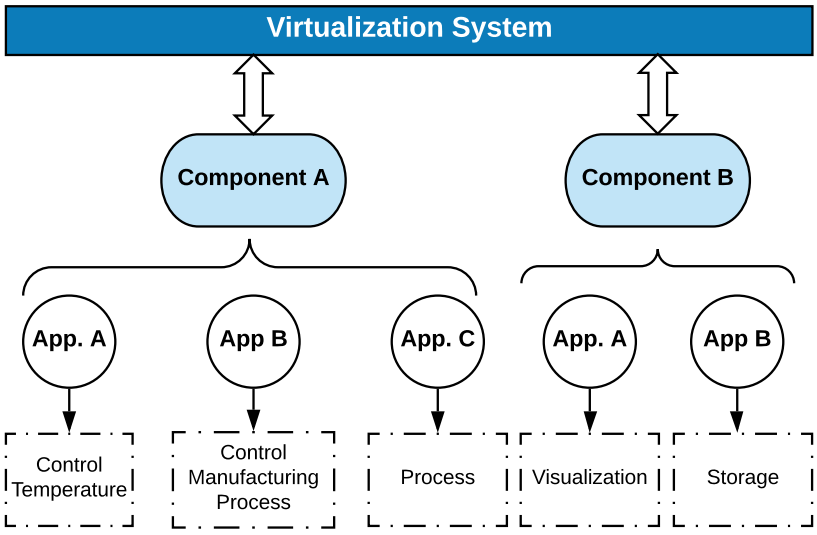


Figure 3.2: IFog4.0 architecture: components and applications

3.2.2 Fog-Management

The ***fog-management*** component is developed to manage Docker containers (components) on the IFog4.0 platform. Although some solutions exist for handling Docker containers (e.g., Portainer), the reason for using a customized solution is to easily integrate new features with higher flexibility since it is not linked to an already-structured product.

The Fog-Management tool allows for installing, running, and stopping applications from a simple and user-friendly dashboard. It also guides the user by adding new components by uploading the corresponding Docker images to the platform. The fog-management tool uses the Docker SDK [132] to deploy and launch a new container. The SDK provides access to the Docker APIs that allow for instantiating, running, and terminating a specific container. As a result, the command will execute the component on the iFog4.0 platform. The end-user can access the part that may create and run new applications inside the container.

In addition, the Docker containers are used to increase the security of the proposed fog architecture; the reason is that the containers are designed not to have a direct connection to the Internet. Moreover, Docker ships with an integrated firewall that blocks access to the internal network; the fog-management tool is the only part that is connected to an external network, which can connect to the Internet to download and install new features and components. The security of the fog-management tool is further hardened by using a Linux firewall. As a result, all major components are stable and protected from Internet attacks.

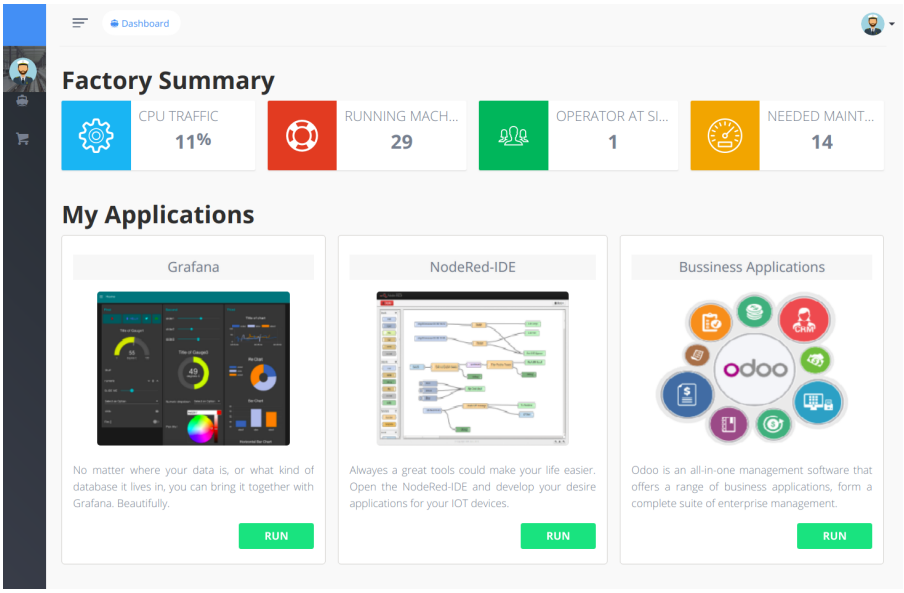


Figure 3.3: Fog-management for components and applications

As shown in Fig. 3.3, three main components are already installed on the IFog4.0 platform: data visualization, programming tools, and ERP. Further, some quick information is provided on the first page, such as the platform and factory statuses on top of the dashboard (i.e., the *CPU usage*, the *Running Machine*, and *Maintenance* statuses).

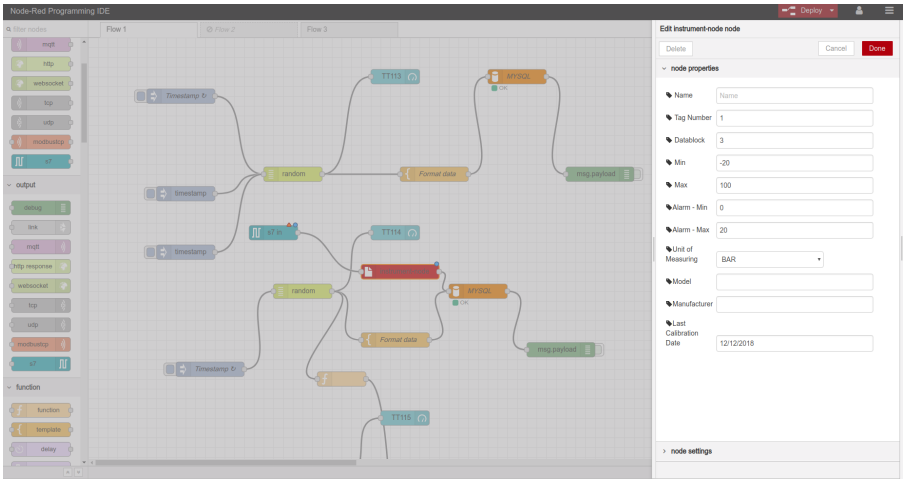


Figure 3.4: Node-RED (fog-programming tools)

3.2.3 Programming tools

The programming tool is implemented based on ***Node-RED*** for visual programming tools [133]. Node-RED was originally developed by IBM Emerging Technology as an open-source product under Apache License Version 2.0 [134] and is now part of the JS Foundation [135]. It provides a flow-based programming (FBP) environment similar to the Function Block Diagram (FBD) programming based on the IEC61131-3 standard. Node-RED provides many black-box nodes with inputs and outputs, with custom programming logic for each node.

The node receives data by its input and processes the input according to the node's logic. The nodes are developed using NodeJs, which is a JavaScript scripting language [133]. In the IFog4.0 architecture, Node-RED is used as a programming tool, and three nodes have been developed as drivers for popular industrial tools: pressure node, differential pressure node, and temperature node. As shown in Fig. 3.4, nodes are developed to manage the instrument data, such as *minimum* and *maximum* alarm thresholds, *instrument type*, *measurement unit*, and *last calibration time* for predictive maintenance purposes.

3.2.4 Data visualization

Grafana is employed for data visualization in the IFog4.0 architecture. Grafana supports many data sources [136]; it can connect to the MySQL data source and store the Node-RED application data in the MySQL database. The PLC Siemens S7-1200 data are stored in the database with the developed node.

3.2.5 Enterprise resource planning(ERP)

Enterprise resource planning (ERP) is an Industry 4.0 component. It helps factories to manage their resources, and it reduces stockroom management costs. Moreover, it helps the industry manage its production request, and it plans to produce a product based on their need. The ERP is one of the essential components to digitalize Small, Medium, and Enterprises (SME). In contrast, IFog4.0 provides an industry 4.0 platform for factories to facilitate industry 4.0 developments. Therefore, according to the RAMI 4.0 model, each factory requires the enterprise resource planning (ERP) system in the business application layer. For this reason, ERP has been included in the IFog4.0 platform alongside other applications. *Odoo* is utilized for the business application layer, and it includes ERP, customer relationship management (CRM), billing, accounting, asset management, and an inventory management system [137]. This platform has a community edition, which is open-source; although, it is possible to add commercial plugins.

3.2.6 Data storage

Data storage is always a key aspect of the edge computing paradigm and fog architecture. MySQL and MongoDB databases are used in this proposal to store data in the relation and non-relation databases. Further, the proposed fog platform provides an easy way to deploy new database systems by adding the Docker image into the fog-management system.

3.2.7 Industrial communication

The industrial sector requires the industrial communication protocol to exchange data in the factory layer. The IoT platform can help the factory to exchange industrial protocol data to the upper layer; however, the IoT platforms usually do not support the industrial protocol. Thus, the gateway plays an essential role in the factory by converting the industrial protocol to IoT protocol (data marshaling). For this purpose, gateways are being built by researchers and companies to convert data from industrial protocols to IoT protocols [32].

As shown in Fig. 3.1, an industrial communication driver is implemented in the utilized Linux kernel. The proposed architecture works with standard industrial protocols, such as Profinet, Modbus, and OPC-UA. All components described in this section are customized and modified for industrial application.

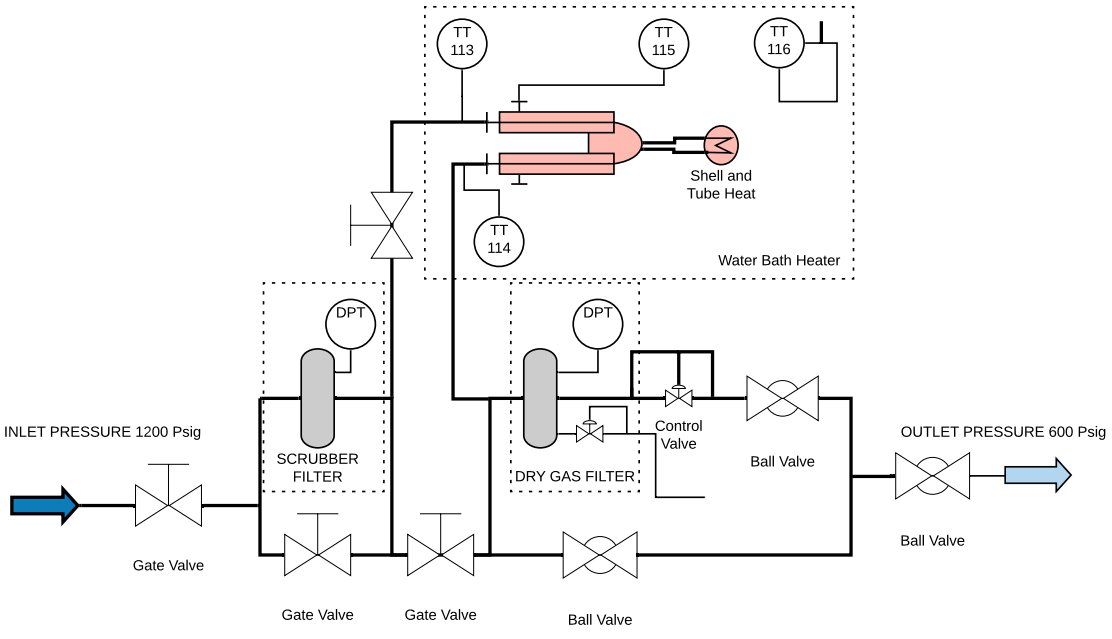


Figure 3.5: Pipe and Instrument Diagram (P&ID) for gas regulation station use case

3.3 Use Case and Results

This chapter explains a proof of concept for the IFog4.0 architecture. For this reason, the proposed IFog4.0 is developed and deployed on a real-life scenario in a factory shop floor to provide Industry 4.0 advantages for SMEs. All technologies utilized in IFog4.0 are based on existing open-source tools, for which additional features are also developed and adopted for industrial applications. The IFog4.0 was developed and emulated using an industrial use case, and it was deployed on a ***gas regulation station***, as described in this section.

The gas regulation station changes the inlet pressure from a higher to a lower amount with the regulators. It has two-stage filters—a *scrubber* and a *dry gas* filter—which are designed for big and small particles, respectively. Fig. 3.5 shows the Pipe and Instrument Diagram (P&ID) and describes the mechanical system & design in detail. The inlet gas pressure is about 1200 $[Psig]$, which, after the two-stage filtering and regulations, drops to 600 $[Psig]$ by using the control valves in the station.

In this scenario, if the pressure decreases, the temperature drops, and the control valve diaphragm will damage quickly. A water bath heater (WBH) was mounted on the station to increase the inlet gas temperature to 60∼65 ℃. (This configuration is part of the ASME International Standard [138].)

The automation of the dry gas filter is implemented in the proposed fog architecture. The filter absorbs solid particles by using a cartridge filter. By considering high gas flow, there is a differential pressure in the vessel. Depending on how full the cartridge is, the differential pressure may vary between 3 and 15 $[Psig]$. The differential pressure transmitter (DPT), which measures the vessel's differential pressure value in real-time, is integrated with the system.

Two main subsystems are presented in this use case. The IFog4.0 architecture is effectively deployed and experimented with these two subsystems. Two PLCs emulate the subsystems. Fig. 3.6 shows that both the subsystem controller and the subsystem emulators are represented within the PLCs.

z

3.3.1 Testbed hardware

The hardware platform that hosts the IFog4.0 system must provide sufficient resources; thus, a computer with a multicore processor and a Gigabit Ethernet network interface card was adopted to deploy IFog4.0; in particular, the PC had an Intel® Core™ i5-6200U CPU @ 2.30GHz x 4 cores. The multicore CPU provided a native parallelization capability. As shown in Fig. 3.7, the two subsystems in the use case were simulated using the same PLCs. In this prototype, two S7-1200 PLCs were used to simulate and emulate the use case, and the D-Link 1 Gigabit Ethernet switch was used to communicate between the PLC network and

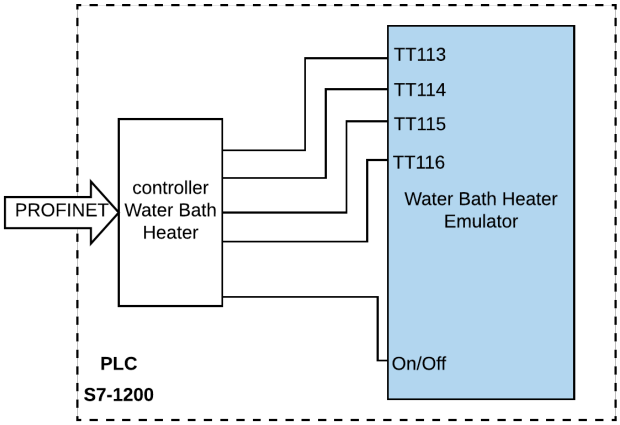


Figure 3.6: Water bath heater (WBH) PLC workflow

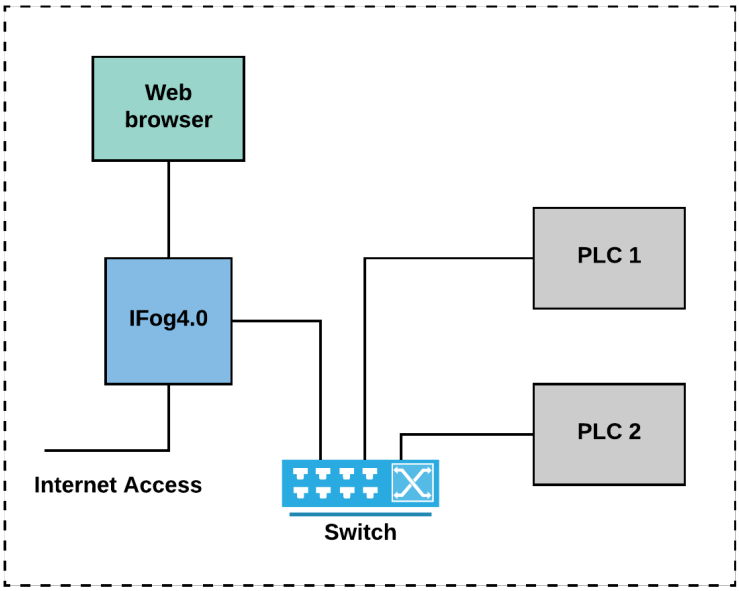


Figure 3.7: Test Configuration

the IFog4.0 platform.

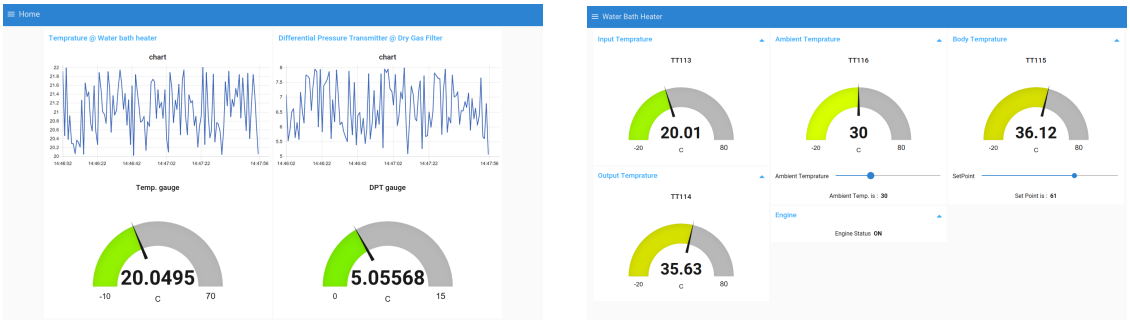
After the hardware was set up, IFog4.0 was deployed on the hardware. The installation process was simple, and the steps were as follows:

- Download Ubuntu Linux server image (the platform is based on the 16.04 LTS version).
- Install Docker container engine and the custom Industrial Communication kernel driver (version 4.15) to provide real-time features.

- Deploy the IFog4.0 main components, available on the IFog4.0 as the Docker containers.
- Install the custom fog-management tool, provided by IFog4.0.
- After running the system, the Docker first runs, then the fog-management tool.
- Then the fog-management system was accessible via the Web interface, available at *http://<IFog4.0 IP address>:8000*.

After accessing the IFog4.0 management system, a login page was available. Once logged in, the system's status was present in the dashboard, and shortcuts to the three main IFog4.0 components—*Grafana*, *IDE (Node-RED)*, and *ERP (Odoo)*—are on the main page. (The industrial communication and database services were working in the background process, and they were not represented in the dashboard. The IFog4.0 repository is still in a private repository but will be disclosed after the first IFog4.0 beta version release.)

Node-RED was utilized to develop an automation application in the gas station. As mentioned in the previous section, ***instrument driver*** is implemented within Node-RED to communicate with the PLCs and IFog4.0 applications. Node-RED has an intuitive drag-and-drop interface that provides tools to design and develop a new program quickly. The ***UI*** node provides a Web interface with a graphical interface that was used to implement the GUI in our use case. As shown in Fig. 3.5, the ***WBH*** had four temperature sensors, which were connected to the node application as inputs. The automation application decided whether to turn on or off the heater on the ***WBH*** subsystem.



(a) Summary for gas regulation station (b) Automation for the WBH system

Figure 3.8: The application developed for the user interface

3.3.2 IFog4.0 installation & configuration

For automating the process, a decision-making algorithm was developed to control the temperature of the gas in the outlet. The PID controller was used to automate the system: it collected the inlet and outlet temperature values and checked the WBH body temperature. The PID controller decided to turn on the engine according to a setpoint value on the WBH. As presented in the diagram, there was an ambient temperature sensor that read the outside temperature. When the ambient temperature was high, the controller would power off the system, as there was no need to use the WBH when the ambient temperature was high, as the control valve will not cause further damage in higher temperatures.

The dry gas filter was the second subsystem. The main goal of this subsystem was to automate the maintenance of the cartridge filter (filter cleaning procedure). As previously explained, the differential pressure increases when the filter is full; hence, a differential pressure sensor measures a higher pressure between the vessels and then decides to open the drain valve when the sensor's value is higher than the set point. Fig. 3.8 shows a UI interface implemented for the dry gas filter and WBH subsystems; this interface is composed of a set point regulator for the outlet gas temperature WBH and an online graph for real-time data stored in the MySQL database.

The **IFog4.0** platform introduced in this chapter is a scalable architecture based on open-source resources, such as *Docker* and *Node-RED*. Open-source components make it possible to achieve greater accessibility, and the IDE tool embedded in iFog4.0 was specifically tailored for this purpose. The architecture also enables component development by using the Docker SDK and deploying it on the platform using the fog-management tool. The proposed workflow guides the end-user to develop a custom application for their needs and install it on their system.

Chapter 4

Wireless Sensor Network: Testbed and Experimental setup

The testbed and experimental setup in Chapters 5 and 6 are described here. As we discussed in the first chapter, the WSN suffers from unpredicted Wi-Fi traffic, and we performed experiments in the presence of external Wi-Fi traffic in this chapter, and we utilized the results to analyze the TSCH network in the following chapters. We performed experimental evaluations with two testbed configurations: the first experiment set was performed to analyze the behavior of the 6TiSCH protocol when channel hopping was disabled, and the second experimental set was obtained when channel hopping was enabled. This experiment aimed to analyze the impact of external Wi-Fi traffic on the channel-hopping mechanism. Different configurations were selected for TSCH matrices (e.g., TXT_RETRIES and N_SLOTEFRAME) to analyze the impact of the 6TiSCH protocol in the presence of external Wi-Fi traffic.

4.1 Experimental testbed

The goal of the experiment was to investigate the effectiveness of the 6TiSCH protocol. Therefore, the hardware which supports the 6TiSCH protocol was selected. For simplicity, at the beginning of the experiment, the star network topology with single-hop links was used in the WSN topology, then extended to the multi-hop network to propose the multi-hop mathematical model. The two devices were worth enough to obtain the impact of the initial configuration of the 6TiSCH protocol and perform the analysis to obtain the network's results and develop a good mathematical model for reliability, latency, and energy consumption. Further, to generate a manageable interference spectrum, four Wi-Fi adapters were generated controlled background traffic on different Wi-Fi channels. The hardware's energy consumption was carefully measured by connecting the hardware to an oscilloscope,

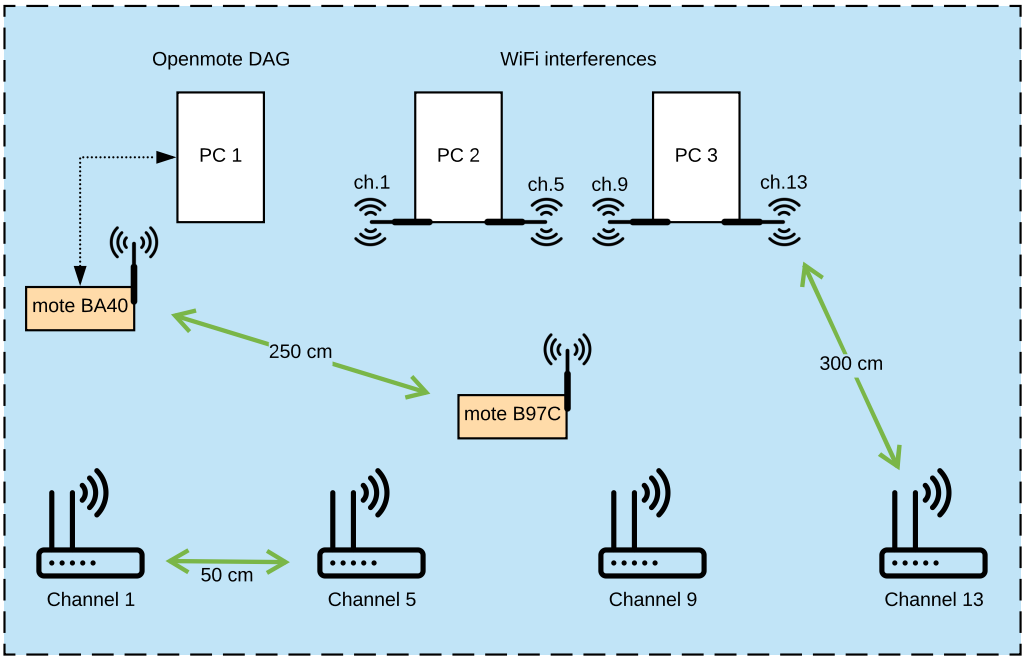


Figure 4.1: Testbed configuration

and the actual value of energy consumption was obtained in each 6TiSCH slotframe (i.e., TX send, RX receive, TX BroadCast).

The OpenMote B hardware was selected to deploy the 6TiSCH protocol in WSN. The device was developed in 2018, and the hardware is designed with a TI CC2538 System on-chip microcontroller. The hardware has a SOC with ARM Cortex TM M3 CPU with 32 KB dynamic RAM and 512 KB of flash memory, including a radio transceiver, compatible with IEEE 802.15.4 for transmission in the 2.4 GHz ISMB band. A few sensors, such as temperature and relative humidity, are embedded in the Wi-Fi device. The device is battery-powered as well as USB-powered.

The OpenMote B hardware is supported by many open-source OSs, such as Contiki and OpenWSN. Both OSs support the 6TiSCH protocol, and both are open-source. The OpenWSN OS (version REL-1.24.0) was selected to deploy the 6TiSCH protocol for the WSN single-hop and multi-hop scenarios. All results in this chapter have been obtained by running the OpenWSN OS on the OpenMote B device.

As shown in Fig. 4.1, the testbed is implemented with four Wi-Fi adapters and routers. All routers were set in fixed channels, which were different from each other. We selected channels 1, 5, 9, and 13 to generate the Wi-Fi traffic with different adapters. In addition, custom software was implemented to generate controlled Wi-Fi traffic over the network. As shown in Fig. 4.1, the adapters were hosted on two computers (PC 2 and PC 3), where, PC 1 controls the Wi-Fi software by sending

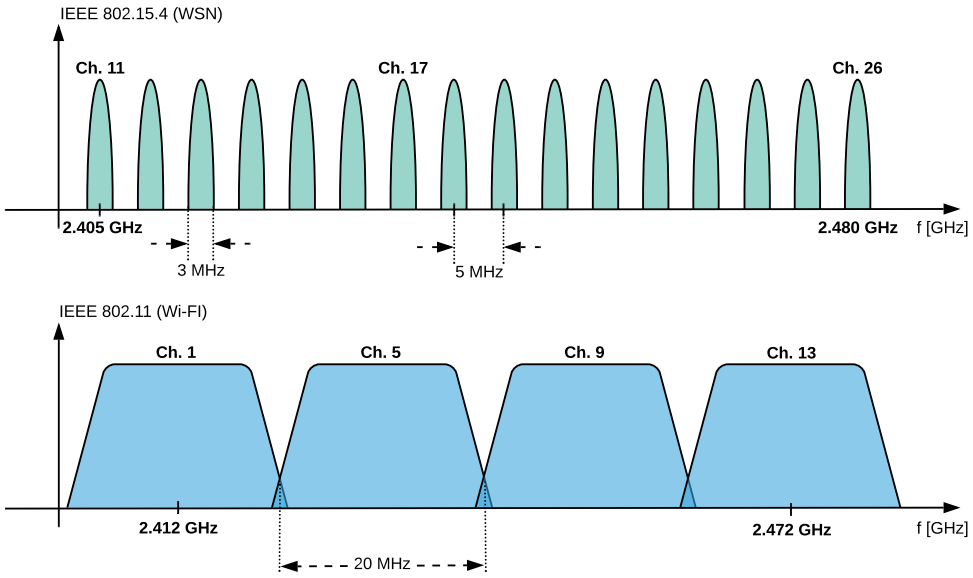


Figure 4.2: IEEE 802.15.4 ISM band vs. IEEE 802.11 OFDM

the start/stop command to PC 2 or PC 3 via the network. The main program that performed the experimental analysis was developed in PC 1, and we utilized the `PING` utilities to send a request to the motes to obtain the round-trip values. The main program sent start/stop commands as needed to the Wi-Fi software, which was on PC 2 and PC 3, and controlled the background traffic. PC 1 sent a configurable ping command to the motes and stores the result in the file.

4.2 Wi-Fi Interference

The Wi-Fi interferences were generated on different channels. Channels 1, 5, 9, and 13 were selected to send the tunable traffic. The harsh environment was emulated with four IEEE 802.11 stations on PC 2 and PC 3, and each PC hosted two adapters. The Wi-Fi stations were placed between two OpenMote B devices to inject the controlled traffic into the air. However, the background traffic generated by the Wi-Fi networks deployed in the offices and labs nearby could also be added to our controlled Wi-Fi traffics.

Additionally, the controlled traffic was injected to all 16 channels in the motes spectrum (i.e., 2.405 to 2.480 GHz) defined by IEEE 802.15.4 in the ISM band. The OFDM channel in the IEEE 802.11 Wi-Fi bandwidth was 20 MHz, so the Wi-Fi channels overlapped with the motes channel by tuning four access points (AP) on

channels 1, 5, 9, and 13, where the channel frequencies were $ch_1 = 2.412\,\text{GHz}$, $ch_5 = 2.432\,\text{GHz}$, $ch_9 = 2.452\,\text{GHz}$, and $ch_13 = 2.472\,\text{GHz}$. As Fig. 4.2 shows[1], the IEEE 802.11 spectrum collided with the IEEE 802.15.4 spectrum. Therefore, it is possible to inject controlled traffic in each mote channel using this configuration. The experiments provide data to analyze the channel hopping effectiveness compared to channel fixed mechanics by considering Wi-Fi traffic.

As explained, the software generates controlled traffic and injects it into the air in a different channel. The software was developed in [69] and implemented based on a finite state machine, which has two states: ***active*** and ***inactive***. A sequence of packets are generated in the ***active*** state. The number of packets within the burst is chosen randomly, based on a truncated exponential distribution. The packets are generated periodically, and the period is $400\,\mu\text{s}$, with packet size 1500 B. After the **active** state, it goes to new states, where the interfering station stops transmitting for a random time whose duration follows a truncated exponential distribution with a mean of 560 ms. This state is the ***inactive*** state. The maximum duration of the ***inactive*** state is limited to 20 s. The software generates traffic with two ***active*** and ***inactive*** states via a dedicated Wi-Fi adapter.

4.3 OpenWSN OS

The OpenMote B hardware supports different OSs, including the OpenWSN . The OpenWSN was developed based on the recent 6TiSCH protocol stack. It is a fully open-source project. It has many advantages; for example, it is possible to check how some features of 6TiSCH are implemented in the OS, and it is easy to modify some parameters in the 6TiSCh protocol and add a new feature in the stack level. For instance, the channel-hopping mechanism was disabled to perform and obtain experimental data for Chapters 5 and 5. This was possible by setting the variable `ieee154e_vars.singleChannel` to the desire channel, which was defined in the source code of the OpenWSN file locate in

`openstack/02a-MAClow/IEEE802154E.c`

to select a fixed transmission channel.

After changing the source code, the code must be compiled, and new executable code should be downloaded into the mote. The OS and all possible applications were compiled from the source code with the cross-compiler on the hosted PC. The cross-compiling software was based on the ARM `gcc` toolchain. OpenVisualizer management software was used to download new executable code to the motes via a USB interface, which was provided along with the OpenWSN platform.

[1]The figure is presented before in this chapter (Fig. 1.4)

The following changes were applied to obtain the experimental results for the next chapters:

- Enable/disable Channel Hopping
- Modify the total N_{slot} in the 6TiSCH matrices
- Modify the maximum number of allowed attempts, N_{tries}
- measure the power consumption value for each slotframe

To modify the N_{slot} and N_{tries} values, the source code located in

`openstack/02b-MAChigh/schedule.h`

and

`openstack/02a-MAClow/IEEE802154E.h`

were modified. The value of the N_{slot} and N_{tries} were changed for various configurations, and the experimental result was obtained to evaluate the resistance of the 6TiSCH network with respect to the Wi-Fi traffic. The analysis of the network is presented in Chapters 5 and 6.

4.4 Measurement

We obtained the measurement to analyze TSCH behavior in the WSN, using the request-response paradigm, like CoAP, by measuring a packet round trip. The conventional `PING` utility was used to perform the experiment, and the `PING` values were stored in the files. The results were analyzed according to the proposed mathematical model to estimate the *reliability* and *power consumption* values. The bash script was developed to perform the experiment and send the command to the other PC, where the Wi-Fi adapters were hosted, to generate and inject controlled traffic into the air. Two sets of experiments were performed, with channel hopping *enabled* and *disabled*. For each set, different values of the N_{slot} and N_{tries} were applied. The values were in $11 \leq N_{slot} \leq 201$ and $2 \leq N_{tries} \leq 15$, respectively. The experiment was carried out on the combination of these values, then the analysis was performed to compare the proposed mathematical model with real data.

The experiment was performed by the `ping` utilities, which is just a straightforward command, `ping6 -s 30 -i 120 -c 10 bbbb:0:0:0:12:4b00:18e0:b97c`, where the first value after **`-s`** specifies the amount of data to be sent in bytes, defined as 30 in the experiment, which sent 38 ICMP data bytes (8 bytes being the ICMP header data). The argument after *-i* defined the waiting interval in seconds. The interval was set to 120 seconds to prevent the communication buffers of the motes from filling up in the case of prolonged interference; this value was set between 30

and 120 to avoid the queuing phenomenon. The third argument was *-c* which defines the total number of pings and stops after sending all pings. These values were different in the experiments, and they depended on the total number of pings in each experiment. The bash script was developed to perform the results in parallel, controlling the Wi-Fi software on other PCs (PC_2 and PC_3).

Statistics about the success or failure of every `ping` request were collected and logged in a file for every experiment. For successful requests, the round-trip time was captured as well, which coincided with d_i values. From the logs, the number N_{L} of failed requests was subsequently computed. This value permitted evaluating the empirical two-way loss probability $\hat{P}_{\mathrm{L}}^{T}$, defined as the measured fraction of requests for which no response is received during the experiment, $\hat{P}_{\mathrm{L}}^{T} = N_{\mathrm{L}}/N_{\mathrm{sam}}$.

Also, the experimental data was stored and published as an open-source dataset in the IEEEDataPort [139].

Chapter 5

Single-hop WSN: Modeling and Performance Analysis of IEEE 802.15.4 TSCH

The work described in this chapter was originally presented in [122].

The WSNs suffer from disturbance and interference generated by co-located Wi-Fi networks near the networks. Hence, it is essential to have a mathematical model to analyze and estimate such a system's behavior in the field. TSCH is used in many industrial applications characterized by demanding reliability and determinism requirements, such as controlling air conditioning, smart lighting, robots, and other applications that need to be controlled remotely by a centralized system. The TSCH protocol's advantage is that it can change the transmission frequency on every attempt, which increases reliability in WSNs.

The performance indicators analysis is essential in WSN's domain. The analysis gives us essential information about the WSN behavior. Moreover, it tells us how much the initial configuration satisfies the application requirements. The Single hop topology is the most common topology in WSN, and it is interesting to propose a method to estimate and analyze the behavior of the single-hop WSN network. This allows us to better understanding the network and quality of the communication.

We developed a mathematical model to estimate and analyze the quality of the communication in the single-hop WSN in this chapter. The model was built by making simple assumptions about the time and frequency diversity's effectiveness. The model was obtained by analyzing the round-trip communication value between two nodes (single-hope). The proposed model analyzes the impact of the network parameters, such as the retry transmission and the number of time slots. Likewise, the effectiveness of channel hopping was analyzed to exploit the ability to prevent narrowband interference from disrupting communication.

5.1 Two-way communication model

We analyzed the behavior of a TSCH communication between two adjacent motes from a theoretical perspective in the presence of unpredictable phenomena. The disturbance and interference were generated from devices not belonging to the network. Further, the network topology settled and the transmission schedule in each slotframe. This transmission scheduling was defined and configured on motes, and collisions no longer occurred in typical operating conditions; therefore, the intra-network interference behavior will not be considered in our analysis. The single-hop communication model was built by starting from TSCH operations and the channel error model.

5.1.1 Packet loss on a single hop

There are many techniques for increasing reliability; the most common is retransmission, in which motes send the same packet over the air more than once. Thus, if the packet fails, another attempt will be transmitted. A failed packet means the packet is sent to the air, but the ACK related to the packet is still not received. In such an event, the MAC layer automatically repeats the acknowledge message over the air up to r_{L} times, where the *retry limit* r_{L} is defined as *macMaxFrameRetries* in the TSCH protocol. Therefore, the packet loss means that all retransmission $(r_{\mathrm{L}} + 1)$ attempts are failed. Further, clear channel assessment (CCA) is an option in the TSCH network, and when the channel is detected as busy, it defers a transmission attempt. This condition is almost the same as a lack of an ACK.

In both methods, when there is a packet failure, the retransmission will be scheduled for the next slotframe, and the total retransmission counter increases by one. Nevertheless, retransmission does not occur immediately in TSCH. In fact, the transmitter must wait for a link (either dedicated or shared) targeted to the destination device.

In our analysis, the single transmission attempts were assumed as a Bernoulli trial model with failure probability ϵ. This hypothesis provided a rough approximation of the actual channel behavior. However, the TSCH network attempts setup was far enough away, so the time diversity sensibly decreased the statistical dependence between them.

If a single dedicated link was allocated between any pair of motes in the slotframe, the transmission attempts in the related slots were placed in intervals by the slotframe time. The slotframe duration is about 2 s, which is longer than the default MSDU lifetime in IEEE 802.11 (i.e., the maximum time after which the transmission process is terminated for a frame in Wi-Fi , including the retries). Also, a random exponential backoff mechanism is applied to avoid collisions. This technic increases the time between retransmissions (retries), and the behavior was

confirmed during the experiment with the real devices.

A random exponential backoff mechanism was also employed for shared links to prevent collisions, which further enlarged the time between retries. The above behavior was verified for the actual devices used for the experiment. Moreover, the subsequent attempts were sent on the same physical channel for the same packet in the channel hopping, which meant that the frequency diversity was exploited. Diversity techniques make retries statistically (almost) independent, so the *packet loss ratio* P_{L} on the link can be calculated as

$$P_{\mathrm{L}} = \epsilon^{r_{\mathrm{L}}+1}. \tag{5.1}$$

where the $r_{\mathrm{L}} = 15$, 20% of failed attempts for $P_{\mathrm{L}} = 6.55 \times 10^{-12}$, and when the failure probability grows up to 50%, $P_{\mathrm{L}} = 1.526 \times 10^{-5}$.

5.1.2 Failure rate for two-way communication

The ping command is considered to perform the request and response transaction to validate the proposed mathematical model with experimental data. As discussed in Chapter 1.2, the *Ping* utility is used to perform the experimental analysis. This tool operates on the Internet Control Message Protocol (ICMP) [140], and generates the same packet for send and receive data in both nodes, for the root and the leaf. In this case, the same mote (the root) is in charge of sending the experimental measurement. Also, the `ping` behavior is the same as that queried via that CoAP protocol.

Every transaction generated by the ping command is sent by two *echo messages*, one for a *request* and one for a *response*. This means that the original mote sends one echo message request to the target mote, then the target mote replies an echo message to the original mote. In our analysis, the single-hop paths were analyzed, and our model was built by considering this assumption; indeed, this assumption is like a star network topology in WSN, meaning one packet is sent from the root to the leaf in the *downward* direction for the request message, and the leaf replies to the root mote in the *upward* direction.

The P_{L}^{T} is the probability that a request-response transaction fails, which means that the short *two-way loss ratio* corresponds to

$$P_{\mathrm{L}}^{T} = 1 - \left(1 - P_{\mathrm{L}}^{D}\right)\left(1 - P_{\mathrm{L}}^{U}\right), \tag{5.2}$$

where P_{L}^{D} is the packet loss probability for request messages, and P_{L}^{U} is the packet loss probability for response messages. Considering that the motes adopt the same radio module and block, the downward and upward links suffer from the same failure probability (P_{L});thus, P_{L}^{T} can be rewritten as

$$P_{\mathrm{L}}^{T} = 1 - (1 - P_{\mathrm{L}})^{2} = 2P_{\mathrm{L}} - P_{\mathrm{L}}^{2}, \tag{5.3}$$

Table 5.1: Glossary of quantities

Quantity	Description	Value
N_{ch}	Number of physical channels	16
N_{slot}	Number of slots in a slotframe	101
T_{slot}	Duration of a slot	20 ms
T_{slfr}	Period of the slotframe	2.02 s
r_{L}	Max. number of MAC retransmissions (retry limit)	15
ϵ	Failure probability for single attempts	-
P_{L}^{x}	Packet loss ratio in direction $x \in \{D, U\}$	-
P_{L}^{T}	Two-way loss ratio	-
P_{r}^{x}	Probability to perform r retries in direction $x \in \{D, U\}$	-
P_{r}^{T}	Probability to perform r retries in both directions	-
N_{sam}	Number of samples per experiment	2880
d_i	Two-way transmission latency of the i-th `ping` request	-
$d_{\text{wait},i}$	Waiting time of the i-th `ping` request	$< T_{\text{slfr}}$
d_{comm}	Two-way network communication time	-
$d_{\text{retr},i}$	Retransmission time of the i-th `ping` request	-
r_i	Total number of retries for the i-th `ping` request	-
D	Two-way transmission latency (random variable)	-
D_{wait}	Waiting time (random variable)	$< T_{\text{slfr}}$
D_{retr}^{x}	Retransmission time in direction $x \in \{D, U\}$ (rand. var.)	-
D_{retr}^{T}	Two-way retransmission time (random variable)	-
μ_d	Two-way mean transmission latency	-
$\hat{\mu}_r$	Estimated mean number of retries (two-way)	-
$\hat{P}_{r}^{T}$	Empirical probability to perform r retries (two-way)	-
$\hat{\epsilon}_P$	Estimated failure probability (from $\hat{P}_{0}^{T}$)	-
$\hat{\epsilon}_D$	Estimated failure probability (from μ_d)	-

and, under our simplified channel error model, it yields to

$$P_{\text{L}}^{T} = 1 - \left(1 - \epsilon^{r_{\text{L}}+1}\right)^2 = 2\epsilon^{r_{\text{L}}+1} - \epsilon^{2(r_{\text{L}}+1)}. \tag{5.4}$$

Applying the same ϵ values (20% and 50%) to this equation produces low values for the two-way loss ratio, which implies that finding a good match between the theoretical model and experimental samples starting from this quantity is barely possible. For example, even when $\epsilon = 0.5$ (which indicates harsh interference), only one packet is lost, on average, every 11 days if the `ping` period is set to 30 s. Further, to prevent packets from remaining queued in the motes' transmission buffers, setting smaller periods in the ping command was avoided during the experiments.

5.1.3 Transmission latency

The proposed model was evaluated with the latencies obtained in the experimental data. d_i is the round-trip delay for the i-th `ping` request, $i \in [1...N_{\text{sam}}]$, where N_{sam} is the total number of samples in the ping test. In the following, *two-way transmission latency* is denoted for simplicity. Three elements are included in d_i:

1. The *waiting time* $d_{\text{wait},i}$ passing from the request packet is queued at the root mote and at the beginning of the downward slot to the target mote. The packet is sent at the earliest opportunity, where the queuing phenomena are not present, which means that the time will be between 0 and T_{slfr} ($0 \leq d_{\text{wait},i} < T_{\text{slfr}}$). The reply packets are directly triggered by requests conveyed in the downward slot in the target mote due to the operations of the ICMP responder process; the responder needs enough time to serve the request and in queue its reply before the beginning of the upward slot.

2. $d_{\text{comm},i}$ is the communication time in the two-way network if there are no failed attempts. This time is measured between the send and received times. It expects that the calculation starts by sending the request packet (downward slot) and stops when receiving the response packet (upward slot). This time depends on the slotframe configuration. The 6TiSCH Operation Sublayer Protocol (6P) in the mote is adopted to manage the slotframe. The downward and upward slots are allocated in fixed locations in the slotframe; consequently, the time between them is fixed, and it is expressed as d_{comm}.

3. $d_{\text{retr},i}$ is the *retransmission time* spent for backoffs (when an attempt is delayed due to a busy channel, which is sent by the CCA function) or retries (when an attempt fails because the frame corrupted due to a collision or noise pulses). $r_i^T = r_i^D + r_i^U$ is the total number of retries, where r_i^D and r_i^U denote the numbers of retransmissions carried out in the downward and upward direction, respectively. Both directions are considered in r_i^T. These quantities depend on background traffic, such as the amount of interference and disturbance in the air. The latency increases by one slotframe when every time packet transmission is retried in each direction. This happens when the network formation has settled—in other words when the slots are dedicated. Finally, the retransmission time depends only on r_i^T and is given by $d_{\text{retr},i} = r_i^T \cdot T_{\text{slfr}}$, where $r_i^T \in [0...2r_{\text{L}}]$.

Overall, the latency of the i-th `ping` request in the two-way transmission is represented as the following:

$$d_i = d_{\text{wait},i} + d_{\text{comm}} + r_i^T \cdot T_{\text{slfr}}. \tag{5.5}$$

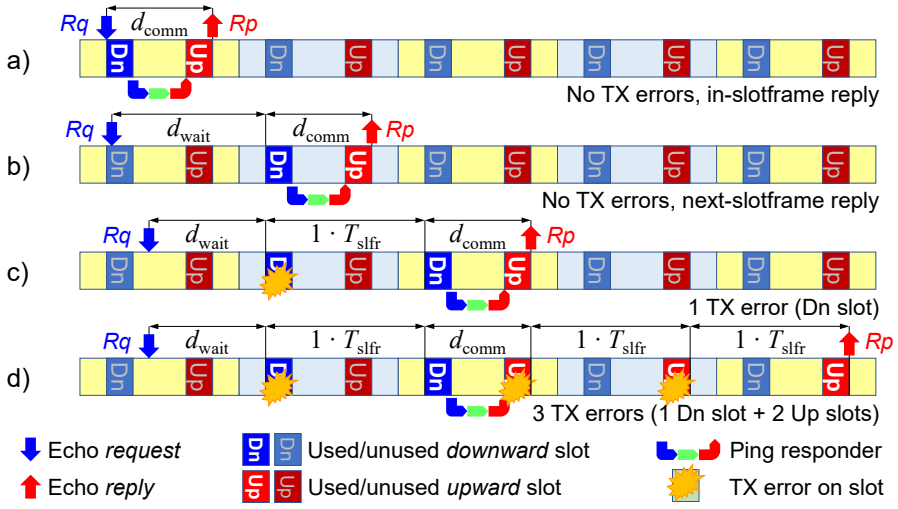


Figure 5.1: Single-hop request-response transaction in TSCH

Fig. 5.1 shows how the above contributions may affect request-response transactions. There are no transmission errors in Diagrams A and B; therefore, no retries were performed. Diagram A showed no waiting time for a reply in this slot (the best scenario), whereas, in Diagram B, the request was received just after the dedicated slot began, so the response had to be sent in the next opportunity. The transmission error use case is shown in Diagrams C and D. The frame was corrupted one or more times in this scenario, meaning some retries were performed. In Diagram C, only the downward link was affected (once), while in Diagram D, the upward link also suffered from errors.

5.1.4 Number of retransmissions

The numbers of retries carried out for a packet on the downward and upward directions can be modeled as random variables, denoted R^D and R^U, respectively. As said above, it can be reasonably assumed that both directions are affected by the same amount and kind of interference. Therefore, R^D and R^U can be considered as independent and identically distributed (i.i.d.) random variables.

For each single direction $x \in \{D, U\}$ let $P_r^x \doteq \mathbb{P}(R^x = r)$ be the probability that a correctly delivered packet underwent exactly r retries (besides the initial attempt). Under our hypotheses about the channel error model, P_r^x can be evaluated as the probability to incur in r failures multiplied by the probability to succeed at the $r + 1$ attempt.

Since latency is only defined for packets whose transmission was successful, the conditional probability given that the packet eventually arrived at destination has

to be considered. This yields

$$P_r^x = \frac{1}{1-P_{\mathrm{L}}}\epsilon^r(1-\epsilon) = \frac{1-\epsilon}{1-\epsilon^{r_{\mathrm{L}}+1}}\epsilon^r. \tag{5.6}$$

Since P_r^x does not depend on the direction x, in the following it will be simply denoted P_r.

When both the downward and upward directions in a two-way request-response transaction are taken into account, the probability $P_r^T \doteq \mathbb{P}(R^D + R^U = r)$ to incur, on the whole, in exactly r retries is, for $0 \le r \le r_{\mathrm{L}}$,

$$\begin{aligned} P_r^T &= \sum_{k=0\ldots r} P_k \cdot P_{r-k} = \sum_{k=0\ldots r} \frac{(1-\epsilon)\epsilon^k}{1-\epsilon^{r_{\mathrm{L}}+1}} \cdot \frac{(1-\epsilon)\epsilon^{r-k}}{1-\epsilon^{r_{\mathrm{L}}+1}} = \\ &= \left(\frac{1-\epsilon}{1-\epsilon^{r_{\mathrm{L}}+1}}\right)^2 (1+r)\,\epsilon^r, \end{aligned} \tag{5.7}$$

where k and $r-k$ are the numbers of retries in the downward and upward directions, respectively, whereas for $r_{\mathrm{L}} < r \le 2r_{\mathrm{L}}$

$$\begin{aligned} P_r^T &= \sum_{k=r-r_{\mathrm{L}}\ldots r_{\mathrm{L}}} P_k \cdot P_{r-k} = \sum_{k=r-r_{\mathrm{L}}\ldots r_{\mathrm{L}}} \frac{(1-\epsilon)\epsilon^k}{1-\epsilon^{r_{\mathrm{L}}+1}} \cdot \frac{(1-\epsilon)\epsilon^{r-k}}{1-\epsilon^{r_{\mathrm{L}}+1}} = \\ &= \left(\frac{1-\epsilon}{1-\epsilon^{r_{\mathrm{L}}+1}}\right)^2 (1+2r_{\mathrm{L}}-r)\,\epsilon^r. \end{aligned} \tag{5.8}$$

In fact, all cases have to be considered where the sum of the numbers of retries in both directions equals r, but no more than r_{L} retries are performed in any single direction. Overall

$$P_r^T = \left(\frac{1-\epsilon}{1-\epsilon^{r_{\mathrm{L}}+1}}\right)^2 (1+\min(r,2r_{\mathrm{L}}-r))\,\epsilon^r. \tag{5.9}$$

5.1.5 Modeling the transmission latency

Since the `ping` request is a two-way transmission, the latency can be modeled as a random variable D. D_{wait} is the waiting time, and it is possible to model in the same way. The time spent in the two directions for retransmission is denoted as D_{retr}^D and D_{retr}^U, respectively. d_{comm} is constant.

As a result

$$D = D_{\mathrm{wait}} + D_{\mathrm{retr}}^D + d_{\mathrm{comm}} + D_{\mathrm{retr}}^U. \tag{5.10}$$

It is possible to consider D_{wait} as uniformly distributed between 0 and T_{slfr} due to the asynchronous cyclic processes in the packet generation and slotframe transmission, whose periods are prime, between 30 s and 2.02 s, respectively. Hence, the probability density function (pdf) is

$$f_{D_{\text{wait}}}(d) = \frac{1}{T_{\text{slfr}}} \left(u\left(d\right) - u\left(d - T_{\text{slfr}}\right) \right), \tag{5.11}$$

where the Heaviside unit step function is $u(\cdot)$. The expected value is equal to half of the slotframe length

$$\mathbb{E}\left[D_{\text{wait}}\right] = \frac{T_{\text{slfr}}}{2}. \tag{5.12}$$

The overall time D^T_{retr} taken by retransmissions corresponds to the sum of the related contributions in both directions, which are i.i.d., and D^T_{retr} is defined as $D^T_{\text{retr}} = D^D_{\text{retr}} + D^U_{\text{retr}}$. Further, starting from P^T_r values, the pdf of D^T_{retr} is

$$f_{D^T_{\text{retr}}}(d) = \sum_{r=0\ldots 2r_{\text{L}}} P^T_r \cdot \delta\left(d - r \cdot T_{\text{slfr}}\right), \tag{5.13}$$

where $\delta(\cdot)$ is the Dirac delta function. For all single-direction $x \in \{D, U\}$, the expected retransmission latency is expressed as

$$\mathbb{E}\left[D^x_{\text{retr}}\right] = \mathbb{E}\left[R^x\right] \cdot T_{\text{slfr}}, \tag{5.14}$$

where

$$\begin{aligned} \mathbb{E}\left[R^x\right] &= \sum_{r=0\ldots r_{\text{L}}} r \cdot P_r = \frac{1-\epsilon}{1-\epsilon^{r_{\text{L}}+1}} \sum_{r=0\ldots r_{\text{L}}} r \cdot \epsilon^r = \\ &= r_{\text{L}} + \frac{1}{1-\epsilon} - \frac{r_{\text{L}}+1}{1-\epsilon^{r_{\text{L}}+1}}, \end{aligned} \tag{5.15}$$

for the known properties of the truncated geometric series. Since $\mathbb{E}\left[R^x\right]$ does not depend on the direction x, $\mathbb{E}\left[R^x\right]$ is denoted as simplified notation $\mathbb{E}\left[R\right]$. Considering these hypotheses, d_{comm}, D_{wait}, D^D_{retr}, and D^U_{retr} are independent values, and D can be calculated from (5.10).

By considering a similar approach described in [141], and using (5.10) and the above pdfs, it is possible to calculate the pdf of latency D as

$$f_D(d) = (f_{D_{\text{wait}}} * f_{D^T_{\text{retr}}})(d - d_{\text{comm}}), \tag{5.16}$$

where the operator $*$ expresses the convolution operation, this function is a piecewise constant function, where the first and last intervals have infinite width and zero height. The $2r_{\text{L}} + 1$ is the inner intervals with width T_{slfr} and height P^T_r / T_{slfr}, starting at $d = d_{\text{comm}}$.

Regarding the cumulative distribution function The cumulative distribution function (CDF) of the overall two-way latency D, denoted as $F_D(\cdot)$, consists of a continuous piecewise linear function with $2r_{\text{L}}+2$ knots (see Figs. 5.2 and 5.3), where knot r, $r \in [0\ldots 2r_{\text{L}} + 1]$ is located at coordinates $\langle d_{\text{comm}} + r \cdot T_{\text{slfr}}, \sum_{i=0\ldots r-1} P^T_r \rangle$.

In conclusion, the expected value of the latency is the sum of the expected values of the single contributions.

$$\mathbb{E}[D] = d_{\text{comm}} + T_{\text{slfr}} \cdot \left(\frac{1}{2} + 2 \cdot \mathbb{E}[R]\right). \tag{5.17}$$

5.1.6 Channel hopping

The failure probability ϵ for attempts is assumed not to vary in the above analysis. This assumption is acceptable when the WSNs are operating on a single channel, as in legacy IEEE 802.15.4 WSNs, it is not sure when the channel-hopping mechanism of TSCH is enabled, which keeps the transmission frequency changing.

Suppose the number of physical channels and the number of slots in the slotframe are prime numbers (as it happened in our 6TiSCH configuration, since $N_{\text{ch}} = 16$ and $N_{\text{slot}} = 101$), the actual frequency on which TSCH performs subsequent attempts hops along all channels according to a pseudo-random value. At best, it is possible to assume that the failure probability ϵ_C of every channel C stays constant over time.

The channel hopping is modeled as a truly random process, where all channels are equally likely to be selected. This assumption is valid if channel black-listing techniques are not used [142], [143]. Even though such techniques could be implemented by modifying the *macHoppingSequenceList* table, maintaining coherence among nodes is difficult. For this reason, a standard black-listing mechanism was not included in the TSCH specification.

Thus, the probability P_C for any given channel C to be used is the same and corresponds to $1/N_{\text{ch}}$. Thus, it provides a reasonable approximation, meaning each transmission attempt is modeled as a two-step random trial. First, a channel is randomly selected, which is characterized by a specific failure probability. Then, the transmission attempt is performed under such conditions. The attempt can be modeled as a Bernoulli trial since the two steps are statistically independent, so the *equivalent failure probability* $\tilde{\epsilon}$ is equal to

$$\tilde{\epsilon} = \sum_{C=1\ldots N_{\text{ch}}} P_C \cdot \epsilon_C = \frac{1}{N_{\text{ch}}} \sum_{C=1\ldots N_{\text{ch}}} \epsilon_C = \overline{\epsilon}. \tag{5.18}$$

Furthermore, the same analysis explained above can then be implemented.

5.2 Experimental evaluation

5.2.1 Experimental testbed

As discussed in Chapter 1.2, the testbed was set up based on the real WSN devices, for which the OpenMote B hardware was selected to deploy the WSN. The modeling in this chapter is considered for a single hop. Therefore, the experimental

data relevant to the analysis was archived by utilizing two devices in a star network topology with single-hop links; one mote plays a root role, which is connected to the PC to send the received request, and the other node is a leaf of the root in the start of the topology.

The OpenWSN [144] OS (version REL-1.24.0) was selected for running on the OpenMote B [145] devices. The OpenMote B devices are relatively new and appeared on the market in 2018. They are built with a TI CC2538 System-On-Chip microcontroller, which integrates an IEEE 802.15.4 radio transceiver for transmission in the 2.4 GHz band, and an ARM® Cortex™ M3 CPU with 512 KB of flash memory and 32 KB of dynamic RAM memory. Additionally, a specific Atmel AT86RF215 chip, which was not used in our experimental campaigns, is available for sub-GHz transmission (868/915 MHz). The OpenWSN hardware is also compatible with the Contiki OS.

In this chapter and most of the thesis experiments, cross-compiling was carried out using the ARM `gcc` toolchain. The produced code was then downloaded to the mote through its USB interface by the `scons` software construction tool, which is provided with OpenWSN .

5.2.2 Interfering traffic

As described in Chapter 1.2, the amount of Wi-Fi traffic in the 2.4 GHz band in the laboratory was not under control. This traffic is generated by APs, notebooks, and many other mobile clients located near the laboratory. Therefore, software was developed to generate and inject controlled Wi-Fi traffic into the wireless network. The software generates traffic by a random generation pattern, as detailed in [69], in which two finite-state machines, *idle* and *burst* states, are implemented to manage the traffic. This software is controlled with another application that orchestrates the packet and traffic generation.

5.2.3 Measurement technique

As discussed before, TSCH behavior is possible to analyze with higher-level protocols, like CoAp, and with the request-response paradigm, such as the `ping` utility. As said before, we wish to investigate those cases where short reaction times are sought, e.g., less than a dozen seconds, as opposed to real-world applications based on WSNs, where the period with which motes are probed is in the order of several minutes or longer. A further reduction of timing does not reflect the typical operating conditions of WSNs, since low power consumption is always given precedence, even in cases where good reactivity is demanded. Moreover, we set the `ping` timeout to 30 s. Therefore, the communication buffers of motes were prevented from filling up in the case of prolonged interference. We stored 120 samples for each

hour. Since the analysis was done for one day, we collected 2880 samples for each experiment day.

We stored all `ping` requests statistics regarding success and failure in a file for every experiment. Further, the round-trip time value was obtained and denoted as d_i. From our logs, the number N_{L} of failed requests was computed. This value permitted us to calculate the two-way loss probability $\hat{P}_{\mathrm{L}}^{T}$, defined by $\hat{P}_{\mathrm{L}}^{T} = N_{\mathrm{L}}/N_{\mathrm{sam}}$, where N_{sam} is the total number of samples.

5.2.4 Matching experimental parameters

In this section, we evaluated the behavior of the theoretical model proposed in Section 5.1 against the real testbed. We analyzed all the experimental data, as are the parameters d_{comm} and ϵ calculated from the real `ping` values. In the real scenario, when the amount of interference and disturbance was tolerable, and enough samples N_{sam} were available (e.g., thousands), it was possible to evaluate a reliable estimation of d_{comm}, where the total value of d_{comm} was the minimum among all measured latencies

$$\hat{d}_{\mathrm{comm}} = d_{\mathrm{min}} \doteq \min_{i=1\ldots N_{\mathrm{sam}}} (d_i). \tag{5.19}$$

In the best case, there were no transmission errors for the request and response packets. Further, no initial waiting time was experienced by the originator of the request. This situation corresponds to set $r_i^T = 0$ and $d_{\mathrm{wait},i} \approx 0$ in (5.5), which leads to (5.19). We described two simple and effective methods to calculate the latency distribution and average latency failure rate.

Failure rate from latency distribution

N_r is the total number of `ping` requests in both directions that succeeded after exactly r retries. Therefore, the waiting time $d_{\mathrm{wait},i}$ should be shorter than the slotframe time, $\forall i \in [1\ldots N_{\mathrm{sam}}]$, N_r can be determined by counting the number of samples for which $\hat{d}_{\mathrm{comm}} + r \cdot T_{\mathrm{slfr}} \leq d_i < \hat{d}_{\mathrm{comm}} + (r+1) \cdot T_{\mathrm{slfr}}$.

In practice, this allows us to evaluate the experimental probability $\hat{P}_r^T$, which corresponds to the statistical relative frequency of experiencing exactly r retries on the two-way path, between the DAG mote (root) and the target mote (leaf), which is written as $\hat{P}_r^T = N_r/(N_{\mathrm{sam}} - N_{\mathrm{L}})$.

The failure rate ϵ was not high in all experiments, which means that the `ping` requests did not suffer from any frame losses in either direction. This indicates that $\hat{P}_0^T$ provides a reliable estimate of P_0^T. From (5.7)

$$P_0^T = \frac{(1-\epsilon)^2}{1 - P_{\mathrm{L}}^T}, \tag{5.20}$$

Then, a reliable estimate of the failure probability is

$$\hat{\epsilon}_P = 1 - \sqrt{\hat{P}_0^T (1 - \hat{P}_{\mathrm{L}}^T)}. \tag{5.21}$$

If the number of failed `ping` requests is so low that P_{L}^T cannot reliably be determined from $\hat{P}_{\mathrm{L}}^T$ (in the carried out experiments in which no requests failed), then an adequate estimate of ϵ can be found by numerically inverting the equation

$$\hat{\epsilon}_P = 1 - \sqrt{\hat{P}_0^T} \left(1 - \hat{\epsilon}_P^{r_{\mathrm{L}}+1}\right) \tag{5.22}$$

We obtained this equation from (5.20) and (5.4).

Failure rate from average latency

The other way to write a term for the failure rate ϵ is to obtain the latency's sample mean value, calculated as

$$\mu_d = \frac{1}{N_{\mathrm{sam}}} \sum_{i=1 \ldots N_{\mathrm{sam}}} d_i. \tag{5.23}$$

By assuming that μ_d provides a reasonable estimation of the expected value of the latency $\mathbb{E}[D]$, an alternative and reliable estimate of ϵ can be found by modifying (5.17).

By the linearity of the expected value, it is possible to estimate the expected number $\mathbb{E}[R]$ of retries in any direction as

$$\hat{\mu}_r = \frac{1}{2} \left(\frac{\mu_d - d_{\mathrm{comm}}}{T_{\mathrm{slfr}}} - \frac{1}{2} \right) \tag{5.24}$$

and by rewriting (5.15) as

$$\hat{\epsilon}_D = 1 - \frac{1}{\hat{\mu}_r - r_{\mathrm{L}} + \frac{r_{\mathrm{L}}+1}{1-\hat{\epsilon}_D^{r_{\mathrm{L}}+1}}} = 1 - \frac{1}{\hat{\mu}_r + \frac{1+r_{\mathrm{L}} \cdot \hat{\epsilon}_D^{r_{\mathrm{L}}+1}}{1-\hat{\epsilon}_D^{r_{\mathrm{L}}+1}}}, \tag{5.25}$$

which is possible to solve numerically.

5.3 Results

The experiment was performed by enabling and disabling channel hopping. We performed several experiments with OpenMote B by changing the interference conditions. The experiment was performed using `ping` utility. Each `ping` request was invoked every 30 s for all experiments. Every experiment was performed for 24 hours, so the total number of samples became $N_{\mathrm{sam}} = 2880$.

Table 5.2: Experimental results and estimated parameters; channel hopping disabled and enabled

Channel hopping disabled (also see plots in Fig. 5.2)

		Measured counters / ratios				Measured latencies (ms)			Estimated failure rate			Computed two-way loss ratio	
Exp.		N_{L}	N_0	$\hat{P}_{\mathrm{L}}^{T}$	$\hat{P}_{0}^{T}$	$d_{\min}$	μ_d	$d_{\max}$	$\hat{\epsilon}_P$	$\hat{\mu}_r$	$\hat{\epsilon}_D$	$P_{\mathrm{L},P}^{T}$	$P_{\mathrm{L},D}^{T}$
$\mathcal{I}_{\emptyset}^{(1)}$		0	2286	0.0	0.794	466	1966.00	10723	0.109	0.121	0.108	8.03×10^{-16}	7.02×10^{-16}
$\mathcal{I}_{\emptyset}^{(2)}$		0	2189	0.0	0.760	464	2059.09	11587	0.128	0.145	0.127	1.06×10^{-14}	8.60×10^{-15}
$\mathcal{I}_{6}^{(1)}$		0	1901	0.0	0.660	460	2373.00	11872	0.188	0.224	0.183	4.69×10^{-12}	3.08×10^{-12}
$\mathcal{I}_{6}^{(2)}$		0	1682	0.0	0.584	464	2723.74	14756	0.236	0.309	0.236	1.82×10^{-10}	1.88×10^{-10}
$\mathcal{I}_{6,6}^{(1)}$		0	1092	0.0	0.379	461	3909.81	19755	0.384	0.604	0.376	4.51×10^{-07}	3.25×10^{-07}
$\mathcal{I}_{6,6}^{(2)}$		0	1318	0.0	0.458	466	3399.57	18553	0.324	0.476	0.323	2.88×10^{-08}	2.75×10^{-08}
$\mathcal{I}_{\emptyset}^{(+)}$	•	0	4475	0.0	0.777	464	2012.55	11587	0.119	0.133	0.118	3.05×10^{-15}	2.69×10^{-15}
$\mathcal{I}_{6}^{(+)}$	•	0	3583	0.0	0.622	460	2548.37	14756	0.211	0.267	0.211	3.16×10^{-11}	3.01×10^{-11}
$\mathcal{I}_{6,6}^{(+)}$	•	0	2410	0.0	0.418	461	3654.69	19755	0.353	0.541	0.351	1.17×10^{-07}	1.06×10^{-07}

Channel hopping enabled (also see plots in Fig. 5.3)

Exp.		N_{L}	N_0	$\hat{P}_{\mathrm{L}}^{T}$	$\hat{P}_{0}^{T}$	$d_{\min}$	μ_d	$d_{\max}$	$\hat{\epsilon}_P$	$\hat{\mu}_r$	$\hat{\epsilon}_D$	$P_{\mathrm{L},P}^{T}$	$P_{\mathrm{L},D}^{T}$
$\mathcal{I}_{\emptyset}^{(1)}$	•	0	2465	0.0	0.856	1937	3278.97	9197	0.075	0.082	0.076	1.94×10^{-18}	2.44×10^{-18}
$\mathcal{I}_{1}^{(1)}$		0	2133	0.0	0.741	1945	3613.18	14003	0.139	0.163	0.140	4.07×10^{-14}	4.40×10^{-14}
$\mathcal{I}_{1}^{(2)}$	•	0	2320	0.0	0.806	1943	3409.05	11278	0.102	0.113	0.101	2.96×10^{-16}	2.51×10^{-16}
$\mathcal{I}_{5}^{(1)}$		0	2481	0.0	0.861	1941	3263.55	10667	0.072	0.077	0.072	1.01×10^{-18}	1.00×10^{-18}
$\mathcal{I}_{1,1}^{(1)}$		0	2109	0.0	0.732	1940	3621.55	12681	0.144	0.166	0.143	7.04×10^{-14}	5.80×10^{-14}
$\mathcal{I}_{1,5}^{(1)}$		0	1926	0.0	0.669	1940	3859.07	12332	0.182	0.225	0.184	2.96×10^{-12}	3.36×10^{-12}
$\mathcal{I}_{1,5}^{(2)}$	•	0	2149	0.0	0.746	1938	3575.46	11289	0.136	0.155	0.134	2.80×10^{-14}	2.28×10^{-14}
$\mathcal{I}_{1,5,9}^{(1)}$		0	1524	0.0	0.529	1940	4438.65	16942	0.273	0.368	0.269	1.86×10^{-09}	1.53×10^{-09}
$\mathcal{I}_{1,5,9}^{(2)}$	•	0	1848	0.0	0.642	1944	3944.73	12761	0.199	0.245	0.197	1.21×10^{-11}	1.02×10^{-11}
$\mathcal{I}_{1,5,13}^{(1)}$		0	1952	0.0	0.678	1941	3810.58	15999	0.177	0.213	0.175	1.81×10^{-12}	1.61×10^{-12}
$\mathcal{I}_{1,5,9,13}^{(1)}$	•	0	1659	0.0	0.576	1942	4277.65	17288	0.241	0.328	0.247	2.59×10^{-10}	3.85×10^{-10}
$\mathcal{I}_{1,5,9,13}^{(2)}$		0	1768	0.0	0.614	1943	4076.80	13649	0.216	0.278	0.218	4.66×10^{-11}	5.06×10^{-11}
$\mathcal{I}_{1,1,5,5}^{(1)}$		0	1638	0.0	0.569	1945	4316.97	16035	0.246	0.337	0.252	3.56×10^{-10}	5.33×10^{-10}

We reported the statistics from the experimental analysis in Table 5.2. The number of failed `ping` requests (N_{L}) is reported in the left part of the table, and the requests that did not experience any retransmissions are shown in the N_0 column. Also, in relative terms, ($\hat{P}_{\mathrm{L}}^{T}$ and $\hat{P}_{0}^{T}$). These values are reflected by the mean (μ_d), minimum ($d_{\min}$), and maximum ($d_{\max}$) values of the measurements of two-way latency. The values $\hat{\epsilon}_P$ and $\hat{\epsilon}_D$ reported in the table are the estimated values and were obtained from $\hat{P}_{0}^{T}$ and μ_d. The estimate $\hat{\mu}_r$ of the average number of retransmissions, also derived from μ_d, is included in the table. Finally, the P_L^T (i.e., $P_{\mathrm{L},P}^{T}$ and $P_{\mathrm{L},D}^{T}$) reported on the rightmost side of the table were obtained from the model, as presented in Section 5.1, by using the above failure rates.

Table 5.2 represents two types of experiments: the upper part shows the experiment with channel hopping enabled, and the lower part shows the experiments with

channel hopping disabled. The value of $d_{\min}$ changed in two cases (approximately 460 ms and 1940 ms). This was due to restarting the motes in two experiments, thus changing the TSCH matrix each time (enabling/disabling channel hopping requires the code of motes to be modified and rebuilt).

The 6TiSCH Experimental Scheduling Function (SFX) expects 6P cell negotiation to start from a randomly chosen cell set. Therefore, after each restart, a different matrix is obtained for motes in the network. This rebooting network indicates a different relative position of the downstream and upstream slots in the slotframe. Additional experiments were performed that confirmed this behavior.

5.3.1 Channel hopping disabled

We perform the first experiment with channel hopping disabled. Intra-network interference was present while performing the test. Nevertheless, the ability to face narrowband disturbance and interference from nearby Wi-Fi network infrastructures was lost, which means that the communication quality may be affected negatively.

We fixed the mote's transmission frequency on WSN channel 17, while the Wi-Fi network adapters were tuned to channel 6, which means that they overlapped with the motes according to the frequency spectrum presented in Fig. 1.4 in Chapter 1.2. As shown in the table, we selected three conditions during the experimental analysis, denoted as $\mathcal{I}_{\emptyset}$, $\mathcal{I}_{6}$, and $\mathcal{I}_{6,6}$, where zero, one, or two Wi-Fi interferers were active, respectively. Two experimental analyses were performed for every interference condition to provide information for variations of the background traffic. The meaning of the numeric superscript in the $\mathcal{I}_{\emptyset}^{(x)}$ (x in parentheses) refers to the same nominal active interference (e.g., $\mathcal{I}_{6}^{(1)}$ and $\mathcal{I}_{6}^{(2)}$ represents that we performed two times experiments in the same condition). Experiments with aggregate datasets are indicated as superscript (+) (e.g., $\mathcal{I}_{\emptyset}^{(+)}$, $\mathcal{I}_{6}^{(+)}$, and $\mathcal{I}_{6,6}^{(+)}$), and we merged all the samples collected in the experiments. The total number of samples was $N_{\text{sam}} = 5760$.

The $\hat{\epsilon}_D$ and $\hat{\epsilon}_P$ are estimated values for the failure rate obtained by both methods (derived from $\hat{P}_0^T$ and μ_D). The estimations were comparable to each other in both methods. However, relying on the calculated mean latency provided a more reliable estimate because all samples were considered. By increasing the amount of Wi-Fi interference, the failure probability also increases. Nonetheless, we calculate a different failure probability estimation in the same Wi-Fi conditions. For instance, in the $\mathcal{I}_{6,6}$ case, two values were estimated for ϵ—that is, 32.4% and 38.4% (obtained from $\hat{P}_0^T$).

Fig. 5.2 shows the measured cumulative frequency distribution (CDF) by indicating thin black lines. We obtained plots using the experimental data, and which have been highlighted in Table 5.2 as • (solid circle). As described before, the datasets with symbols (+) are merged to overcome variability of the background

traffic (e.g., $\mathcal{I}_{\emptyset}^{(+)}$, $\mathcal{I}_{6}^{(+)}$, and $\mathcal{I}_{6,6}^{(+)}$). The CDFs obtained from the theoretical model are the plots in colored lines. As shown, the experimental data and the theoretical model are identically aligned with each other, which means that the assumptions made on the channel error model do not generate any practical weakness. Further, the experimental and theoretical data are matched in the tail part of the curves, located on the rightmost side of the figure. This implies that the approximation given by the model is satisfactory for all ranges of latency values.

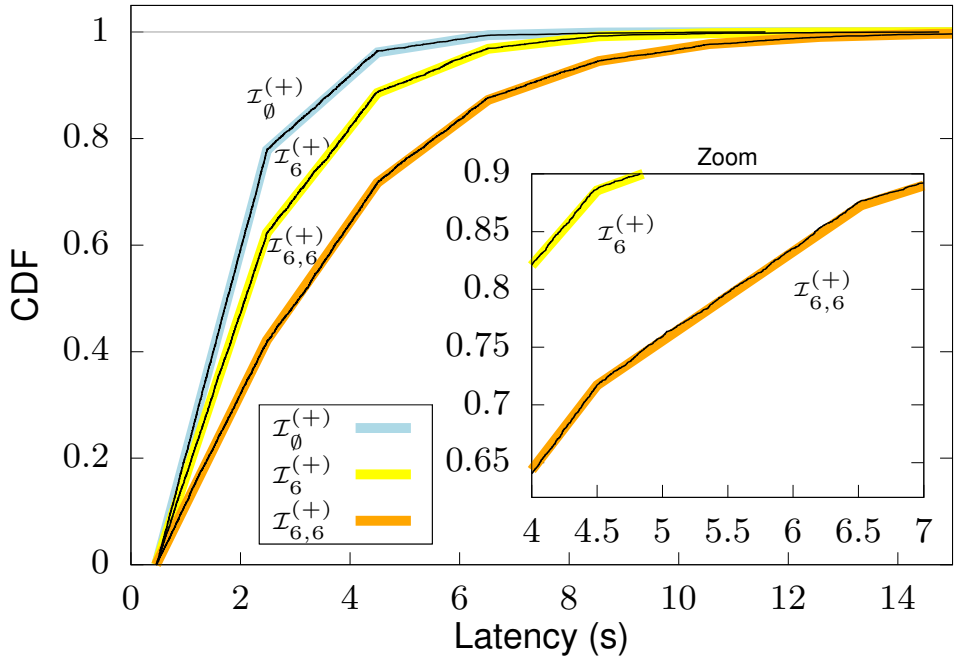


Figure 5.2: Measured and theoretical CDFs of d (channel hopping disabled).

5.3.2 Channel hopping enabled

We performed the second experiment by activating the Wi-Fi interferers up to four stations, and we set no more than two generating traffic on the same channel. The table has the same notation as before, based on Wi-Fi channels on which controlled traffic was injected. The results, reported in the lower part of Table 5.2, do not have many differences with respect to the previous experiment. We calculate the $\hat{\epsilon}_P$ and $\hat{\epsilon}_D$ of the failure probability from the $\hat{P}_0^T$ and μ_d, and the results are the same.

The failure rate is lower than when the channel hopping was disabled; this improvement was expected from the channel-hopping mechanism. The controlled background traffic undermined the repeatability of the experiments, and it allowed only a partially meaningful direct comparison of the results performed at different

times. This appears precisely by matching side-by-side analyses carried out with the same amount of Wi-Fi traffic. For instance, the failure rates ϵ are obtained 19.9% and 27.3% for $\mathcal{I}^{(2)}_{1,5,9}$ and $\mathcal{I}^{(1)}_{1,5,9}$ (derived from $\hat{P}^T_0$), respectively. Thus, by increasing the amount of Wi-Fi traffic, the ϵ worsens, beginning from about 7.5% with no Wi-Fi traffic up to 21.6%–24.6%, with four active interferences (obtained from $\hat{P}^T_0$). The equivalent failure rate achieved by channel hopping when all spectra were occupied with four channels was, on average, lower than the failure rate calculated when channel hopping was disabled. In this case, two identical interferences were tuned to the same motes' frequency.

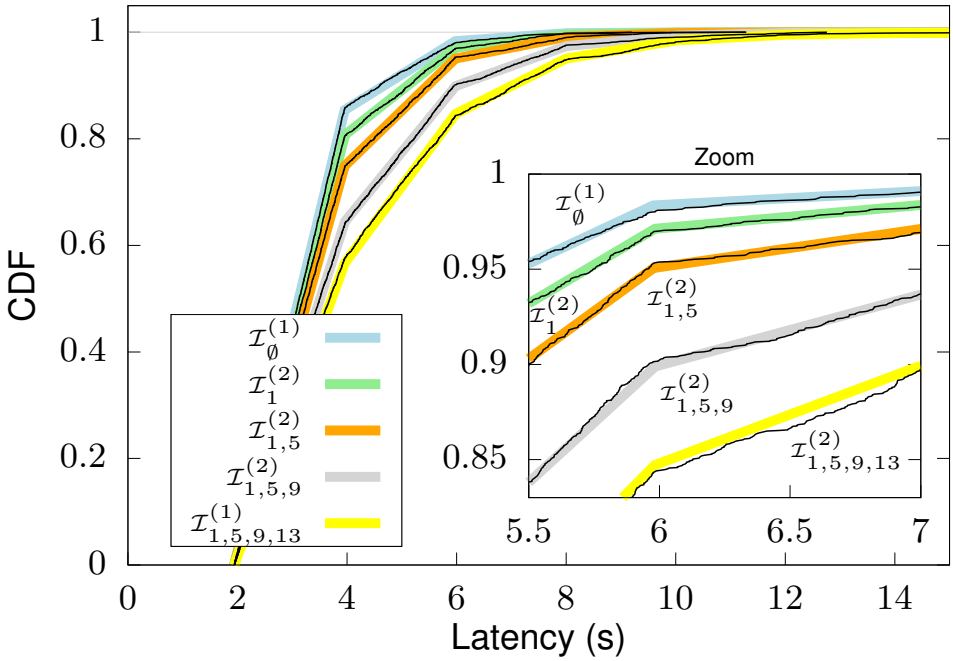


Figure 5.3: Measured and theoretical CDFs of d (channel hopping enabled).

The theoretical and measured CDFs evaluated for five interfering situations when channel hopping was enabled are shown in Fig. 5.3 (labeled with a solid circle in the lower part of the table). Again, the curves show a good match between the analytical model and the experimental results. Additionally, many other experiments were performed by setting the retried parameter to lower values (e.g., $R = 1$, 3). The estimations for $\hat{\epsilon}_P$ obtained varied between 9–14%. Despite the spectrum variability, these values are equivalent to the results in the table. Furthermore, when only a single retry was allowed (which indicated that packet communication was not unlikely to fail), the estimated and measured `ping` failure rates were almost equal ($P^T_L = 1.9\%$ and $\hat{P}^T_{\mathrm{L}} = 1.7\%$, respectively).

5.3.3 Comments on channel hopping effectiveness

As presented in Table 5.2, the quality of communication experienced by motes during the experiments was linked to the mean amount of controlled Wi-Fi traffic injected into the network. For example, the fraction of failed attempts with 4 Wi-Fi traffics tuned on different channels (in case the transmission frequency of motes changed on every attempt) matched the case when the mote's frequency was fixed and only one Wi-Fi traffic was activated within that frequency range. Assuming that the background Wi-Fi traffic on the various channels was the same and did not vary significantly between the experiments, the average interference was roughly the same, regardless of the channel on which their radio communications are tuned at the time of transmission.

This supports the assumption that channel hopping makes communicating motes see an equivalent spectrum that averages all involved physical channels' behavior, as proposed by this equation (5.18), therefore decreasing the variability of the communication quality. In other words, our experimental test validated, to some extent, the ability of TSCH to flatten narrowband interference.

The use of Wi-Fi devices has increased; such devices can be found everywhere, and their diffusion is steadily increasing over time, especially in industry. However, the use of WSNs is essential for Industry 4.0, and it is a critical element of the future of factories. This is why it is essential to have better performance indicators in WSN communication. Further, traffic may impair communication in the WSN, either by causing repeated collisions or delaying transmissions because of the carrier sensing mechanism. The TSCH, especially the channel-hopping mechanism, was purposely proposed to prevent such irregular phenomena.

5.4 Conclusions

In this chapter, experimental analysis was performed on 6TiSCH WSN with a star topology when the number of interfering Wi-Fi traffic was different. The experimental analysis was performed to evaluate the communication quality, such as latency and reliability, where mote devices were utilized to provide a round-trip time with a ping utility, which was used to provide a CoAP-like request-response paradigm. Finally, a simple theoretical model was proposed to describe the single-hop WSN communication network, where trials were considered independent and subject to a time-invariant failure probability. Then, the model was compared with the experimental data store with ping utility (round-trip time) via the OpenMote B .

The experimental analysis validated the ability of TSCH to smoothen narrowband interference and disturbance by providing an equivalent quality of communication for motes. This value was roughly average of what was seen on the various physical channels, which means that the quality of communication between motes

will not be affected much when there are accepted interferences near to motes.

The proposed model is not meant to compare the behavior of TSCH with legacy WSN solutions. In fact, in many of the latter, subsequent attempts cannot be considered statistically independent (which led to the definition of solutions relying on time slotting and channel hopping). In the next chapter, the mathematical model is extended to the multi-hop networks. However, the transmission latencies over multiple hops are expected to grow and become less predictable, making the TSCH approach less appealing for applications subject to specific timing constraints (e.g., to close control loops with slow dynamics).

Chapter 6

Multi-hop WSN: Modeling and Performance Analysis of IEEE 802.15.4 TSCH

The work described in this chapter was originally presented in [146].

This chapter proposes a simple and effective mathematical formulation that describes a TSCH network's behavior, with the ultimate goal of optimizing performance indicators. The model considers both configuration parameters, estimates the protocol's performance indicators and aims to characterize the network and the environment. This model was obtained and validated by applying experimental measurements performed and obtained on a real setup (see Chapter 1.2).

This chapter also proposes an experimental characterization of the power consumption of OpenMote B devices, which helps perform a truthful estimation of the power consumption. However, these devices are widely used in industrial WSN applications, and they only recently support the 6TiSCH protocol. Nevertheless, to the best of our knowledge, this is the first analysis of OpenMote B power consumption proposed in the literature.

The mathematical model is first proposed for the performance indicators of a multi-hop WSN. Then, experiments related to the power-consumption model are presented in Section 6.2. Also, an experimental analysis is provided in Section 6.3. Finally, some practical application contexts concerning performance indicators are reported in Section 6.4.

6.1 Mathematical model

Start topologies (e.g., single-hop) are applied in many industrial applications. An example of this kind of WSN architecture can be found in gas station production lines, in which sensors must send data in high frequencies to centralized control rooms. In this architecture, all sensors are connected to a centralized control unit. Thus, all nodes except the root are leaves and conceptually lie at the same network level. Suppose all network nodes are not at the same level; a proper routing mechanism would then be expected for communication, which relies on packet relays operated by middle nodes as a gateway, according to a multi-hop transmission mechanism.

The advantage of multi-hop solutions is to cover large areas; however, communication latency increases in this method. Multi-hop WSNs usually rely on a multi-level tree topology derived from a root node [147]. Mesh topologies also exist where routes between nodes are arbitrary, but they are less common in practice. Many WSN protocol stacks, like 6TiSCH, rely on RPL to obtain a logical tree topology out of a mesh physical topology. The most used WSN topologies are multi-level networks, as these can easily be adapted to star topologies just by setting the number of the level in the model to one.

The distance between the coordinator (root) and leaf in WSNs is N_{level}, where $N_{level} = 1$ presents the single-hop network. The request/response paradigm is considered for this analysis, as in Chapter 5, such as the CoAP (Constrained Application Protocol), [123], which is a good solution for modern WSNs. Therefore, every communication between two motes consists of a request packet sent in the forward direction, then quickly replied by the packet in the return direction.

The term "packet i" is applied to both the request and response packets to simplify the equations. As a result, the two-way communication is described by setting $N_{hop} = 2$ in star topologies (one hop from the root to the target mote and another level backward, from mote to root). In addition, $N_{hop} = 2 \cdot l$ is the number of hops that are required to query a node at level l. When a leaf node sends a query in a balanced tree, $N_{hop} = 2 \cdot N_{level}$.

In this section, the analysis is divide into two phases. The first phase (Sections 6.1.1, 6.1.2, and 6.1.3) describes the network model and how to adapt it to an actual setup–in particular, an analysis is performed to estimate the performance indicators, and many parameters of the model can be obtained directly from a set of measurements performed on an experimental testbed. In the second phase (Section 6.1.4), the model parameters are obtained directly from the testbed; then, the obtained parameters are applied indirectly to derive new quantities, which helps estimate the expected behavior of a TSCH network when its protocol parameters are varied. The proposed methodology allows us to compare model/protocol parameters and performance indicators and enables a fast network configuration starting from the requirements of each application.

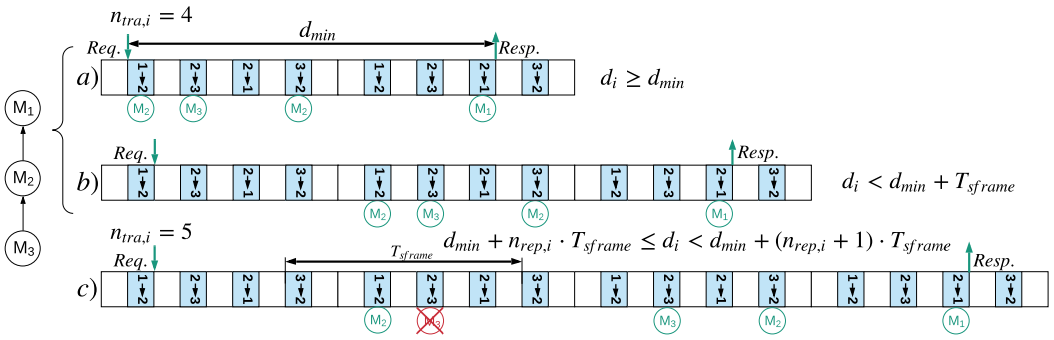


Figure 6.1: The example of request/response iteration in TSCH without (Cases A and B) and with (Case C) transmission errors ($n_{rep,i} = n_{tra,i} - N_{hop}$ represents the overall number of retransmissions performed for the i-th packet on the two-way path).

When model parameters are estimated in the first phase, queuing insides nodes is not considered. In other words we assumed that, for any given mote, only one packet with the same destination for the next hop can be found in the queue at any given time. From our viewpoint, this condition is not very restrictive. In fact, it can be easily met if the packet rate with which measurements are performed is much below the available capacity of the path [148]. It is worth pointing out that, in most real application scenarios, the period of data transmissions on a WSN is quite long, which means that queuing phenomena are typically negligible. In the following analysis, we considered paths with a single dedicated cell for each hop, i.e., the matrix of each mote does not contain more than one cell targeted to the same destination. Again, this can be considered a typical configuration for such matrix. The model quantities used in this chapter are summarized in Table 6.1.

6.1.1 Reliability

Industrial applications mostly have a high demand for *reliability*, which means that all transmitted packets–or, at least, most of them–need to reach the destination. Automatic repeat request (ARQ) schema allows every packet transmission to be retransmitted (upon failures) on each communication link up to the retry limit.

The possibility for any frame to reach the target node is defined as a parameter called *frame error probability* ϵ (i.e., the chance for a single try in the probability that two nodes near each other fail), N_{tries} is the maximum number of permitted tries for each frame at the MAC layer (the total number of retries is $N_{tries} + 1$), and N_{hop} is the number of hops performed at the routing layer, which is considered in both directions).

As a result, starting from the frame error rate and the number of hops on the

Table 6.1: Glossary of Quantities.

Quantity	Description
N_{slot}	Number of slots in the slotframe
T_{slot}	Duration of a slot
T_{sframe}	Duration of the slotframe
N_{tries}	Maximum number of allowed attempts per frame
N_{level}	Maximum distance between a mote and the root
N_{hop}	Number of hops between source and destination motes
N_{sam}	Number of samples collected in the experiment
T_{app}	Generation period of packets at the application level
$N_{tra,i}$	Actual number of frames transmitted to deliver packet i
N_{lost}	Measured number of lost packets
$\hat{N}_{lost}$	Estimated number of lost packets
$\hat{\overline{n}}_{tra}$	Average number of frames transmitted per delivered packet
P_{lost}	Packet loss ratio
ϵ	Frame error probability
ϵ_{pkt}	Packet loss probability
p_{nr}	Probability that no retries are made for a packet
E	Total energy consumption
E^f_{tx}	Energy consumption to transmit a confirmed frame
E^f_{rx}	Energy consumption to receive a confirmed frame
E^f_{listen}	Energy consumption for idle listening (single slot)
$\hat{N}_{tra}$	Total number of frames transmitted in the experiment
f_{tra}	Rate of frame exchanges (including retries)
f_{listen}	Rate at which idle listening occurs
d_i	Latency experienced by packet i
d_{min}	Minimum latency
μ_d	Mean latency
σ_d	Standard deviation of latency
d_{p99}	99-percentile of latency
d_{max}	Maximum latency
Max_d	Theoretical worst-case latency

The hat symbol ($\hat{}$) over a quantity denotes estimated values

two-way path, the *packet delivery probability* can be obtained as

$$1 - \epsilon_{pkt} = \left(1 - \epsilon^{N_{tries}}\right)^{N_{hop}}. \tag{6.1}$$

where ϵ_{pkt} is *packet loss probability* for both the request and the respond directions.

Three assumptions are considered to hold the above equation. First, the upward or downward directions have the same value of ϵ considered for the frame error

probability. This assumption indicates that the radio modules are similar for both motes. Second, the value of ϵ must not change over time. This assumption is not generally true, but the experiments performed on a real 6TiSCH network subject to non-negligible interference produced by several nearby Wi-Fi networks suggest that this approximation is often acceptable [122]. (An experimental analysis and further explanation about this second assumption are described in Chapter 5). Third, the value of ϵ must be the same for all the path links. This assumption is mostly valid for small networks based on a star topology, but, when $N_{level} \geq 2$, it may not be acceptable in some cases due to the amount of disturbance and interference that could vary sensibly over the coverage of the WSN (which for multi-hop configurations can be bored).

Not unreasonably, it can be assumed that the spatial distribution of interfering Wi-Fi devices (and, in general, of equipment exploiting wireless communication technologies) is more or less homogeneous, as is that of the traffic they generate. Indeed, IT administrators place the access points with this goal in mind.

The frame error probability ϵ could be obtained by analyzing the round-trip time measurements. The packet latency is denoted as d_i, which is the packet's transmission time sending the i-th request and the packet's reception time that replies to it. The quantity d_i is just for successful transactions, where both the request and response packets are accurately delivered.

The overall number of transmission attempts performed on-air is denoted as $n_{tra,i}$, where i is the iteration packets involved in any given request/response. All hops considered (in both directions) can be obtained from the duration d_i, using the following term

$$n_{tra,i} = \left\lfloor \frac{d_i - d_{min}}{T_{sframe}} \right\rfloor + N_{hop}, \tag{6.2}$$

In this equation, the minimum communication latency is denoted as d_{min}. This value is evaluated over all the measurements, and it depends on the TSCH matrix setup, where the TSCH slotframe duration may be written as $T_{sframe} = N_{slot} \cdot T_{slot}$.

To better understand the (6.2), an example is shown in Fig. 6.1, which demonstrates different possibilities of the TSCH matrix. In this example, 3 motes are considered, where mote M_1 is the root, and the other motes are children to each other until the last leaf (i.e., M_2 is the child of the root, and M_3 is the child of M_2). The notation $x \rightarrow y$ in a cell of the TSCH matrix denotes a cell reserved for transmission from M_x to M_y. For example, $1 \rightarrow 2$ specifies that the cell is dedicated for transmission from M_1 to M_2, while $2 \rightarrow 1$ indicates that it is reserved for transmission inverse direction.

In the case of a request/response, transmission between the root M_1 and the leaf mote M_3 is performed without errors (i.e., when no re-transmission is employed), $n_{tra,i} = 4$ and latency is in the range of $d_{min} \leq d_i < d_{min} + T_{sframe}$. In particular, when d_i equals (or is slightly greater than) d_{min} (Fig. 6.1.a), this indicates that there is a packet in a queue on M_1 to be sent before those in cell $1 \rightarrow 2$.

Considering that packet generation by applications is not synchronous with the TSCH matrix when the transmission request is issued relative to the slotframe boundaries, it is possible to describe this as a random variable uniformly distributed between 0 and T_{sframe}. In the case of no re-transmissions, the maximum delay is limited by $d_{min}+T_{sframe}$, and is experienced when the packet is queued immediately after the beginning of the $1 \rightarrow 2$ cell, which means that it is sent in the next slotframe in cell $1 \rightarrow 2$ (Fig. 6.1.b). At the end, when there is a transmission error, the d_i latency is gained by the length of one slotframe (T_{sframe}) for every frame retransmission. For example, in the case of a single retransmission being performed (Fig. 6.1.c), on any link, bounds $d_{min} + T_{sframe} \leq d_i < d_{min} + 2 \cdot T_{sframe}$ can be defined for latency. Equation (6.2), which allows for calculating the real number of transmission attempts on-air, is obtained from these terms for latency.

The probability p_{nr} that a packet is correctly exchanged with the minimum number of transmission attempts (i.e., the probability that no retransmissions are tried) can be obtained from our experiments as

$$p_{\mathrm{nr}} \;=\; (1-\epsilon)^{N_{hop}} = \frac{|\{i \mid n_{tra,i} = N_{hop}\}|}{N_{sam}}, \tag{6.3}$$

where $|\{i \mid n_{tra,i} = N_{hop}\}|$ is the number of packets that arrived at the destination after precisely N_{hop} transmission tries (on the whole path), and N_{sam} is the number of samples obtained in the experiment (i.e., the number of request/response repetitions performed by the application).

The frame error probability can be obtained from (6.3), and is written as

$$\epsilon \;=\; 1 - \left(\frac{|\{i \mid n_{tra,i} = N_{hop}\}|}{N_{sam}}\right)^{\frac{1}{N_{hop}}}. \tag{6.4}$$

ϵ is obtained by considering that the error rates on each of the 16 channels are utilized in the communication channel by the TSCH mechanism.

In the case of a multi-level (hop) network, it also "aggregates" the failure probabilities on all links that make up the path between the source and the destination of the request/response exchange, and provides a simple yet effective model of the overall behavior of the wireless spectrum in the place where the WSN is deployed. As previously stated, this approximation is acceptable only if the error rates on the links composing the path do not differ significantly.

6.1.2 Power consumption

Since motes cannot usually connect to a stable external power source and instead must rely on batteries or energy harvesting, the *power consumption* evaluation is essential in WSNs. A simple but effective power-consumption model [124] subdivides the total energy consumption into separate contributions

$$E = E_{tx} + E_{rx} + E_{listen} + E_{cpu} + E_{sleep}, \tag{6.5}$$

where E_{tx} is the overall energy required for transmission (that includes the transmission of data frames and reception of the related ACK frames), E_{rx} is the overall energy required for frame reception (that includes the reception of data frames and transmission of the related ACK frames), and E_{listen} is the energy consumed to listen to the channel when waiting to receive frames, and the related slots remain idle (*idle listening*).

In this chapter, the two contributions, E_{cpu} and E_{sleep}, have been embedded in the same quantity, $E_{comp} = E_{cpu} + E_{sleep}$, which represents the energy consumed by the mote when no specific application that uses the network runs on it. In the following, the focus is only on the energy consumed for communication. However, the measured value of E_{comp} for the motes used in the experimental campaign is listed in Section 6.2.

The energy consumption E_{net} of the network component over a given time interval can be directly obtained from the number n_{listen} of cells reserved for transmission (as per the TSCH matrix) that remain unused, in which receiving nodes uselessly sense the channel and the number n_{tra} of cells in which transmission is actually performed.

$$\begin{aligned} E_{net} &= E_{tx} + E_{rx} + E_{listen} \\ &= n_{tra} \cdot \left(E^f_{tx} + E^f_{rx}\right) + n_{listen} \cdot E^f_{listen}, \end{aligned} \tag{6.6}$$

where E^f_{tx} and E^f_{rx} correspond to the energy consumed to send and receive a single frame (data plus the related ACK), respectively, while E^f_{listen} is the energy consumed to listen to the channel when a reserved cell is not used.

The equation (6.6) can be rewritten in terms of the power P as

$$P = f_{tra} \cdot \left(E^f_{tx} + E^f_{rx}\right) + f_{listen} \cdot E^f_{listen}, \tag{6.7}$$

where f_{tra} is the mean number of verified frame transmission attempts per second performed on-air by all motes (i.e., the mean overall frame transmission rate), while f_{listen} presents the mean number of cells allocated for reception over a time span of one second in which idle listening occurred.

The rate f_{tra} can be calculated by dividing the total number N_{tra} of frames transmitted on-air in the experiment by the overall duration of the experiment itself

$$f_{tra} = \frac{N_{tra}}{T_{app} \cdot N_{sam}}, \tag{6.8}$$

where T_{app} is the period in which requests are repeatedly issued by the originating mote.

Alternately, the rate f_{listen} was achieved by subtracting N_{tra} (i.e., the number of cells that were used in the experiment) from the total number of cells kept

for transmission in the same interval, and then divided by the duration of the experiment:

$$f_{listen} = \frac{1}{T_{app} \cdot N_{sam}} \cdot \cdot \left(\frac{N_{hop} \cdot T_{app} \cdot N_{sam}}{T_{sframe}} - N_{tra} \right), \quad (6.9)$$

which can be rewritten as the slot repetition frequency $1/T_{slot}$ multiplied by the fraction of slots in the slotframe reserved for the transmission of the considered request/response packets, minus the mean overall frame transmission rate f_{tra}

$$f_{listen} = \frac{N_{hop}}{T_{sframe}} - f_{tra} = \frac{N_{hop}}{T_{slot} \cdot N_{slot}} - f_{tra}. \quad (6.10)$$

Number N_{tra} is composed of two contributions: the number N_{tra}^{deliv} of transmitted frames associated with packets correctly delivered to the destinations, and the number N_{tra}^{lost} of transmitted frames associated with packets that were lost

$$N_{tra} = N_{tra}^{deliv} + N_{tra}^{lost}. \quad (6.11)$$

N_{tra}^{deliv} can be computed directly from the $n_{tra,i}$ values inferred by applying (6.2) to experimental samples

$$N_{tra}^{deliv} = \sum_{i=0}^{N_{sam}} n_{tra,i}. \quad (6.12)$$

Instead, the value of N_{tra}^{lost} cannot be evaluated directly in the experimental setup. In fact, latency is not defined for lost packets, which implies that (6.2) cannot be applied. Thus, in the following analysis, estimate $\hat{N}_{tra}$ will be use instead of N_{tra}, obtained by substituting $\hat{N}_{tra}^{lost}$ in (6.11). Note that, in the case of no packets being lost (i.e., when $N_{lost} = 0$), the quantity N_{tra} can be derived directly from experimental data, and the same holds for f_{tra} and f_{listen}.

Equations (6.8) and (6.10) are kept only when the injected traffic does not exceed the allocated network capacity–that is, if $f_{tra} \leq N_{hop}/T_{sframe}$; otherwise, the network behavior is unstable, and the number of packets queued in the motes' transmission buffer will increase indefinitely.

Instead, the equation (6.2), which utilizes the round-trip time d_i measured on the given path to compute the number $n_{tra,i}$ of transmission tries performed for the i-th request, is kept only if packets never experience queuing delay due to previously buffered packets that use the same outgoing cell. Indeed, the same condition also affects all formulas that depend on (6.2). However, as highlighted in Section 6.1.4, the approximation above is acceptable, provided that the traffic injected in the network is low. As disturbance and interference on-air do not depend on queuing, ϵ can be estimated by setting T_{app} for measurements long enough to prevent buffer overrun requirements.

$$\hat{N}_{tra}^{lost} = \hat{N}_{lost} \cdot \sum_{h=0}^{N_{hop}-1} \left[\frac{(1-\epsilon^{N_{tries}})^h - (1-\epsilon^{N_{tries}})^{h+1}}{1-(1-\epsilon^{N_{tries}})^{N_{hop}}} \right] \cdot \left(h \cdot \left(\frac{1}{1-\epsilon} - \frac{N_{tries} \cdot \epsilon^{N_{tries}}}{1-\epsilon^{N_{tries}}} \right) + N_{tries} \right) \tag{6.13}$$

6.1.3 Latency

Latency is of primary importance in soft and firm real-time systems, and it is expected to obey application-dependent deadlines. For this kind of system, the delays due to transmission over a digital network must be bounded, and specific real-time industrial protocols and implementations are normally employed to provide guarantees (on a mathematical basis) for the worst-case transmission times [149], [150]. In TSCH, this can be obtained by utilizing deterministic channel access (i.e., time slotting) coupled with the proper scheduling of exchanges (matrix configuration). The theoretical *worst-case latency* (Max_d), considering that the only source of delay are retransmissions, can be computed analytically as

$$\begin{aligned} \mathrm{Max}_d &= N_{hop} \cdot N_{tries} \cdot T_{sframe}. \\ &= N_{hop} \cdot N_{tries} \cdot (N_{slot} \cdot T_{slot}) . \end{aligned} \tag{6.14}$$

The equation (6.14) holds only if, for every link in the path, no queuing of the packets occurs in any mote. A sufficient requirement for this to happen, in case all cells reserved for the request/response exchange are not utilized by other traffic, is that $T_{app} \geq \mathrm{Max}_d / N_{hop} = N_{tries} \cdot T_{sframe}$.

Conversely, while queuing phenomena occur, Max_d can be computed by multiplying the value in (6.14) by the maximum size of the local queues (considering the packet under transmission). From (6.14), it is clear that the worst-case latency can be reduced by lowering N_{slot} (which increases power consumption) or by lowering N_{tries} (which decreases power consumption but worsens reliability).

6.1.4 Derived quantities

In this subsection, perspective changes are proposed by laying the ground for a comprehensive analysis of TSCH-based WSNs. A complete characterization of the network performance can be obtained starting from the experimental results in terms of performance indicators. First, the idea is to estimate ϵ from a set of experimental data by (6.4). Several intermediate quantities are then derived, which permit the expected network behavior to be modeled and evaluated analytically. Besides ϵ, which describes to what extent disturbance and interference negatively impact communication, the quantities that affect TSCH operation depend on:

- the characteristics of the protocol decided in the network design/configuration phase (N_{slot}, T_{slot}, and N_{tries});

- the characteristics of the path, which mainly depend on the physical placement of nodes and obstacles (N_{hop});
- the characteristics of the hardware of motes in terms of power consumption (E^f_{tx}, E^f_{rx}, and E^f_{listen});
- the characteristics of the software application that communicates over the network (T_{app} and N_{sam}).

One of the difficulties with the previous analysis is that the measured number N_{lost} of failed requests in typical operating conditions may not represent disturbance and interference reliably. Unless the experiments are deferred for long (months), no packets are usually lost with the default retry limit. A more reliable estimate of the number of failed requests can be derived from (6.1) and corresponds to

$$\hat{N}_{lost} = N_{sam} \cdot \epsilon_{pkt} = N_{sam} \cdot \left(1 - (1 - \epsilon^{N_{tries}})^{N_{hop}}\right). \tag{6.15}$$

The expected number of transmission attempts performed for a packet correctly delivered on a single hop is described by a truncated geometric series

$$\frac{1}{1 - \epsilon^{N_{tries}}} \sum_{k=1}^{N_{tries}} k(1-\epsilon)\epsilon^{k-1}.$$

Concerning the whole path, it can be easily demonstrated that an estimate $\hat{n}_{tra}$ of the average number of frames transmitted on-air for a successful request/response exchange can be calculated as

$$\hat{n}_{tra} = N_{hop} \cdot \left(\frac{1}{1-\epsilon} - \frac{N_{tries} \cdot \epsilon^{N_{tries}}}{1 - \epsilon^{N_{tries}}}\right). \tag{6.16}$$

An estimate $\hat{N}^{lost}_{tra}$ of the overall number of frame transmission attempts associated with failed requests can be obtained from (6.13), where the factor in square brackets represents the fraction of packets that are lost in hop $h+1$ (that is, for which the preceding h hops were successful). In contrast, the factor in round brackets represents the average number of transmitted frames for each one of these packets (the first term resembles (6.16), while, for the latter, when a packet is lost because its transmission fails on a particular link, the number of attempts on that link equals N_{tries}).

An estimate of the rate f_{tra}, as defined by (6.8), can be obtained from the derived quantities $\hat{N}_{lost}$, $\hat{n}_{tra}$, and $\hat{N}^{lost}_{tra}$, and corresponds to

$$\hat{f}_{tra} = \frac{\hat{n}_{tra} \cdot (N_{sam} - \hat{N}_{lost}) + \hat{N}^{lost}_{tra}}{T_{app} \cdot N_{sam}}. \tag{6.17}$$

By substituting $\hat{f}_{tra}$ in (6.10), the estimate $\hat{f}_{listen}$ can easily be computed. In turn, the power consumption P is obtained from $\hat{f}_{tra}$ and $\hat{f}_{listen}$ by (6.7). The quantities E^f_{tx}, E^f_{rx}, and E^f_{listen} for the commercial motes utilized in the testbed were evaluated experimentally.

Lastly, since the overall number of frames transmitted on-air for a single end-to-end packet exchange equals N_{hop} when no errors occur, and every retry uses an additional slotframe, an estimate of the average transmission latency (that, for request-response pairs, coincides with the round-trip time) can be obtained from $\hat{n}_{tra}$ and d_{min} as

$$\hat{\mu}_d = d_{min} + \left(\frac{1}{2} + \hat{n}_{tra} - N_{hop}\right) \cdot T_{sframe}. \tag{6.18}$$

Practically, equations (6.15)–(6.17) are valid also in the presence of queuing phenomena. The only limitation is that they do not consider packets that are lost because a receiver's queue (either an intermediate node or one of the endpoints) is full at the time of arrival. This situation rarely occurs in real cases, and its occurrence is indirect evidence that the network was incorrectly sized.

In contrast, neglecting queuing delays, as in (6.18), constitutes an acceptable approximation only when (most of the time) packets manage to be forwarded at the earliest opportunity (i.e., in the next suitable link). Nevertheless, if the average delay experienced by packets in the queues of the traversed motes along the path can be estimated, Equation (6.18) can be updated to consider this latency contribution.

After computing the value of ϵ by (6.4)[1]), all the performance indicators we are interested in for describing a WSN based on TSCH from the point of view of applications–namely, reliability ($1 - \epsilon_{pkt}$), power consumption (P), theoretical worst-case latency (Max_d), and average latency ($\hat{\mu}_d$)–can be obtained analytically using Equations (6.1), (6.7), (6.14), and (6.18), respectively.

6.2 Power-consumption Model

In this section, the power consumption of the specific motes used in the experimental campaign was measured experimentally. Two main goals were accomplished: first, the real power consumption of OpenMote B devices was assessed; second, the actual values of E^f_{tx}, E^f_{rx}, and E^f_{listen} were obtained. By substituting these values in (6.7), an estimate of the actual power consumption of this kind of mote can be obtained, given the network traffic and TSCH configuration parameters. All experiments were performed on OpenMote B devices [145]. An experimental campaign

[1]The actual value of d_{min} can also be derived from the configuration of the TSCH matrix (see the example in Fig. 6.1.

was carried out on real OpenMote B devices for the four above configurations, using the network setup described in Section 4.

6.2.1 Characterization of power consumption

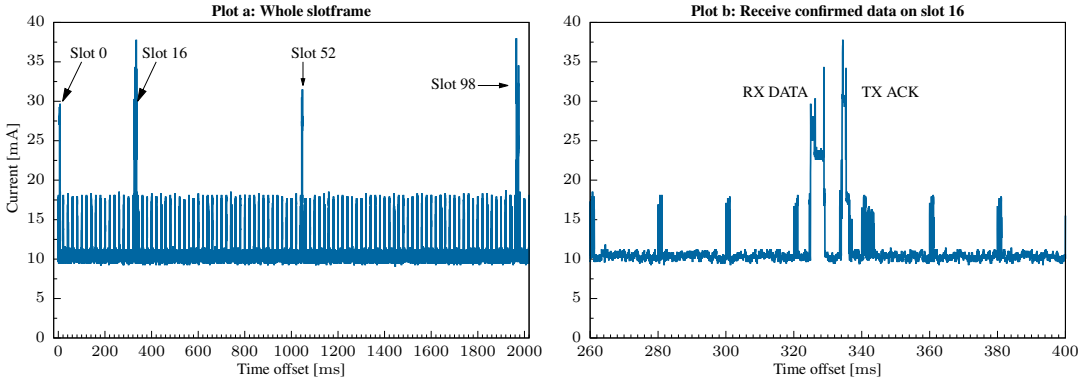


Figure 6.2: Power consumption for the duration of the slotframe (Plot 6.2.a) and for a zoomed-out portion of it that embraces seven slots (Plot 6.2.b). In the plot on the right side, the reception of a confirmed frame (bearing a `ping` request) can be observed in the fourth slot.

OpenMote B motes' power consumption was estimated using a Tektronix MDO3024 oscilloscope equipped with two TPP0250 probes. The probes were connected to two specific pins of the mote's printed circuit board dedicated to measuring the total current flow it absorbs. Current consumption was measured only on the leaf mote, which is the one that, in real applications, is powered by batteries. The OpenWSN code was modified to disable the onboard LEDs, as this led to a reduction of the absorbed current, equal, on average, to about 6 mA, corresponding to about 360 μJ saved for every slot.

The experimental measures of the current drained by the leaf mote over one slotframe are reported in Fig. 6.2a, where the x-axis represents the offset (in milliseconds) from the beginning of the slotframe. The many regularly-spaced short peaks are related to the protocol stack's activities at the beginning of every slot. Consequently, their number is equal to $N_{slot} = 101$. Fig. 6.2b zooms out part of the diagram in Fig. 6.2a and includes seven peaks, which occur every $T_{slot} = 20\,\mu$s. The two current consumption patterns labeled *RX DATA* and *TX ACK*, which rise above the previous peaks, refer to a confirmed transmission (associated with a `ping` request/response exchange) as seen by the point of view of the leaf mote that manages the RX cell.

For all experiments presented in this section, the cells in the TSCH matrix were configured as follows:

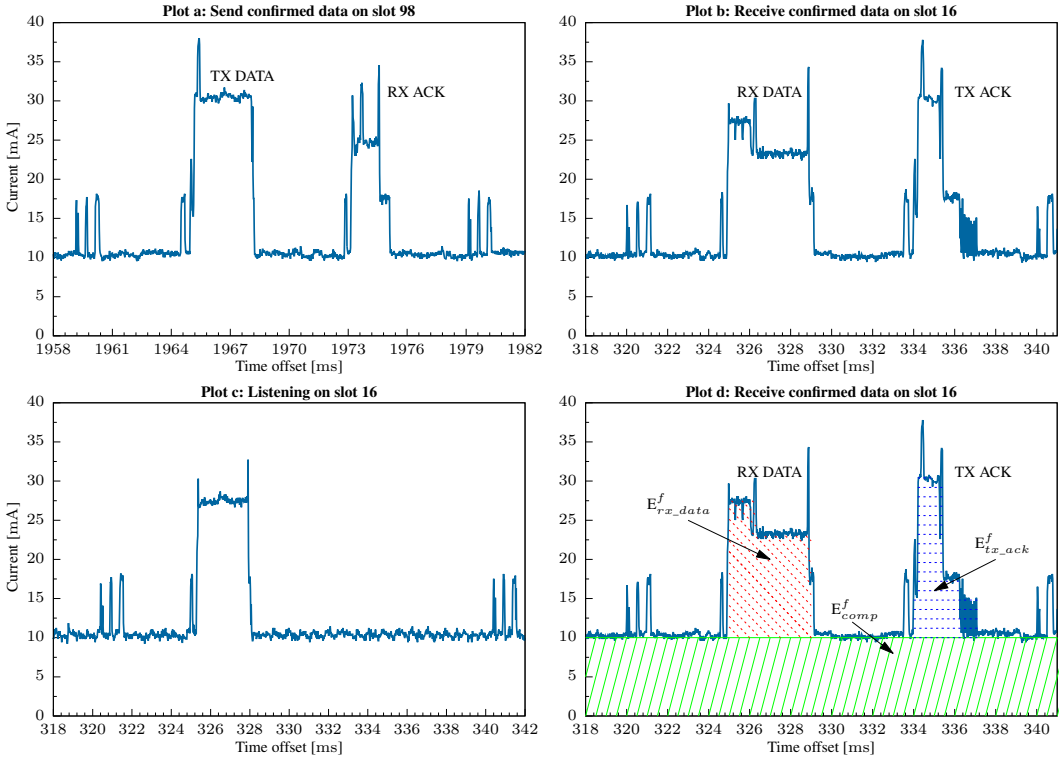


Figure 6.3: Power consumption for the different types of cells: slot including a confirmed frame transmission (a), slot including confirmed frame reception (b), slot in which idle listening occurs (c), and dissection of a confirmed frame reception into separate contributions (d).

- Slot 0 (time offset 0 ms) was a shared TXRX cell, which is typically associated with channel offset 0. It is used to send frames related to network formation and maintenance (e.g., enhanced beacons).
- Slot 16 (time offset 320 ms) was configured in the root as a shared TXRX cell, while from the responder view, it behaves like a dedicated RX cell. After the initial network configuration phase, the root is typically used only to send confirmed packets to its children in the downward direction; hence, collisions may not occur. In this context, this cell was used for transferring `ping` request packets. An example that shows what happens to current absorption in the responder during the reception of a frame bearing the `ping` request and the transmission of the related ACK frame can be found in Fig. 6.3b. When no frame is transferred and the cell remains unused, the responder still expends a considerable amount of energy listening to the channel (idle listening), as shown in Fig. 6.3c.
- Slot 98 (time offset 1960 ms) was configured in the responder as a non-shared

Table 6.2: Energy consumption for different types of actions within a slotframe matrix with OpenMote B motes. In **bold** quantities used in Eq. (6.7).

Quantity	Action(s)	Slot offset	Energy [μJ]	Size [bytes]
$E^f_{rx\_data}$	RX DATA frame	16	178	87
$E^f_{tx\_ack}$	TX ACK frame	16	106	33
$E^f_{tx\_data}$	TX DATA frame	98	187	90
$E^f_{rx\_ack}$	RX ACK frame	98	79	33
E^f_{listen}	Idle listening	16	**138**	-
E^f_{comp}	Computation	-	628	-
E^f_{rx}	RX DATA + TX ACK	16	**284**	120
E^f_{tx}	TX DATA + RX ACK	98	**266**	123

TX cell and was reserved for transmission to its parent (the root) in the upward direction. In our experiments, this cell was used for transferring packets bearing the `ping` response, an example of which is shown in Fig. 6.3a.

- Slot 52 (time offset 1040 ms) was configured in the responder as a shared TXRX cell and was devoted to communication with its children. However, as this mote in our setup was a leaf (i.e., it has no children), the cell remained unused in the experiment.

Experimental results related to power consumption are reported in Table 6.2. They were obtained by performing a numerical integration on the experimental samples included in the above plots (plus other samples not reported for space reasons) and by multiplying the resulting area, which refers to the overall electric charge in micro coulombs (μC), by the supply voltage of the mote (3 V), to obtain the power consumption in microjoule (μJ).

In particular, the quantity E^f_{comp} refers to the power consumption when no transmission or reception is performed within the slot (i.e., the mote's global consumption, excluding operations of the network component). Such a quantity corresponds to the rectangular area at the bottom of Fig. 6.3d, filled with a pattern of continuous green lines, and, for a time slot lasting 20 ms, it is about 628 μJ. The column *Slot offset* specifies the placement of the activity in each time frame; for example, the $E^f_{rx\_data}$ happens after 380 ms ($20 * 16$) for a duration of 20 ms after starting the slotframe in each network cycle.

This value is noticeably higher than for other kinds of motes. For instance, in [125] a similar analysis was performed for motes based on the STM32F103RB 32-bit microcontroller and the Atmel AT86RF231 radio chip. In this case, the power

consumption (obtained by multiplying the charge in microcoulomb, as reported in the paper by 3 V) was about 113.4 μJ. The abnormally high power consumption obtained in the setup depended on the fact that OpenMote B devices, in several respects, are prototypes, and the software they run (OpenWSN) is not yet optimized for energy-saving (e.g., by switching the CPU to deeper sleep states when no operations are needed).

The other quantities reported in the table refer to the network component only. For instance, the values of $E^f_{rx\_data}$, and $E^f_{tx\_ack}$ were obtained through numerical integration of the areas filled with red dashed lines and blue horizontal dashed lines in Fig. 6.3d, respectively. Starting from the consumption related to the DATA and ACK frames, the quantities $E_{rx} = E^f_{rx\_data} + E^f_{tx\_ack}$ and $E_{tx} = E^f_{tx\_data} + E^f_{rx\_ack}$ were obtained, which, for a confirmed frame exchange, corresponded to the consumption on the receiver side (RX DATA + TX ACK) and on the transmitter side (TX DATA + RX ACK), respectively.

To emulate application-level request/response exchanges (as those performed by CoAP), the `ping` utility was used. In all the experiments, the size of the payload included in `ping` packets was set to 30 bytes. These packets were encoded using the rules of the IEEE 802.15.4 standard, which lead to the frame sizes reported in Table 6.2 (which also includes the physical preamble). Since, in our setup, we considered the responder (leaf node), E_{rx} is associated with an ICMP *echo request* packet, while E_{tx} is associated with an ICMP *echo reply* packet.

6.3 Results

Two sets of experiments were performed to analyze the most relevant protocol parameters' effects on performance indicators; in particular, this section evaluates the impact of the variation of TSCH matrix parameters, such as the slotframe length and the maximum number of allowed tries.

The experimental WSN was comprised of two nodes (OpenMote B devices) and was the same as the setup used for evaluating power consumption in Section 6.2. This overly simplified configuration did not limit the validity of our results. Concerning data exchange, the TSCH matrix permits the partition of the whole traffic into independently managed cells, according to a known and configurable schedule. The only additional complexity when dealing with multi-level networks is that, due to the wider coverage of the network, the frame error probability ϵ on specific links may differ, and (6.4) provides only an aggregated value for it. The `ping` generation period (that corresponds to T_{app}) was set to 120 s, and every experiment lasted 4 hours. Therefore, the number of samples collected in each experiment was $N_{sam} = 120$.

Four interfering IEEE 802.11 stations were located near the two OpenMote B devices to put the OpenMote B in a harsh environment and inject a controllable

amount of traffic into the air. The testbed set up is explained in detail in Section 4.

6.3.1 Performance vs. slotframe length

From the first set of experiments, the impact of the number N_{slot} of slots in a slotframe on the performance indicators is analyzed in Section 6.1. In particular, this parameter was varied between 11 and 201 (the default value in OpenWSN is $N_{slot} = 101$). Motes were restarted at the beginning of every experiment, as modifying N_{slot} requires the OS to be recompiled and reloaded on every mote, which produces a new network formation and a new TSCH matrix. For this reason, each experiment was characterized by a different value of d_{min}. The other network parameters were kept at their default values, and, in particular, $N_{tries} = 16$.

Results of the experimental analysis are reported in Table 6.3. As expected, smaller values of N_{slot} improve network responsiveness: if $N_{slot} = 11$, the measured maximum latency was 1.438 s; whereas for $N_{slot} = 201$, it increased to 14.050 s (the theoretical worst-case values were 7.04 s and 128.64 s, respectively). On the other hand, when $N_{slot} = 11$, power consumption was about 150 times higher than when $N_{slot} = 201$ (1262.8 μW and 76.7 μW, respectively). This means that, concerning the selection of N_{slot}, a compromise had to be made.

Table 6.3: Experimental results about the influence of N_{slot} on latency, reliability, and power consumption (measured on real devices).

	Latency								Reliability			Power Consumption				
N_{slot}	d_{min}	μ_d	σ_d	d_{p99}	d_{max}	$\hat{n}_{tra}$	$\hat{\mu}_d$	Max_d	P_{lost}	ϵ	$1-\epsilon_{pkt}$	f_{tra}	f_{listen}	P	$\hat{f}_{tra}$	$\hat{f}_{listen}$
			[s]			[#]	[s]	[s]				$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$	μW	$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$
11	0.212	0.409	0.194	1.231	1.438	2.34	0.399	7.040	0.0	0.148	12-nines	2.00	90.70	1262.8	1.95	90.71
31	0.491	0.982	0.431	2.301	3.419	2.27	0.969	19.840	0.0	0.119	14-nines	1.91	32.06	453.0	1.89	32.06
51	0.258	1.024	0.649	3.007	3.054	2.25	1.021	32.640	0.0	0.110	15-nines	1.88	19.41	278.3	1.87	19.42
91	0.497	1.741	1.046	4.861	5.397	2.25	1.858	58.240	0.0	0.110	15-nines	1.87	10.80	159.4	1.87	10.80
101	0.352	2.046	1.588	8.764	10.457	2.28	1.936	64.640	0.0	0.124	14-nines	1.95	9.70	144.7	1.90	9.71
151	2.877	5.036	1.755	8.846	14.557	2.25	5.135	96.640	0.0	0.110	15-nines	1.85	6.44	99.0	1.87	6.43
201	0.726	4.216	2.880	12.131	14.050	2.36	4.193	128.640	0.0	0.153	12-nines	1.97	4.78	76.7	1.97	4.78

The reason for the above behavior is that, although the actual rate f_{tra} of frame transmissions on-air did not vary appreciably for the different experiments, there was an increase in the rate of f_{listen} with which motes listened to the network for receiving frames. f_{tra} remained stable because it depends on the (fixed) packet generation period at the application level and the quality of communication on the wireless medium (i.e., the frame error probability ϵ). As expected, the values of ϵ estimated in the different experiments were not the same. The traffic injected by the interfering Wi-Fi nodes was, on average, fixed, but the load caused by other Wi-Fi

devices located close to the motes was out of our control and varied unpredictably. The values obtained for ϵ ranged from 11.0 % to 15.3 %. Despite this variation, because of the high value set for the retry limit, the measured *packet loss ratio* $P_{lost} = N_{lost}/N_{sam}$ was equal to 0 in all experiments.

Values for $\hat{\mu}_d$, $\hat{f}_{tra}$, and $\hat{f}_{listen}$ in Table 6.3 are computed starting from ϵ and derived quantities (that is, not measured directly in the experiments), and can be used to cross-check the proposed model. Although our experiments included only 120 samples, they were very close to the corresponding measured quantities (i.e., μ_d, f_{tra}, and f_{listen}). The unit of measurement of $\hat{f}_{tra}$, $\hat{f}_{listen}$, f_{tra} and f_{listen} are kHz in all experimental analyzes.

6.3.2 Performance vs. retry limit

In the second set of experiments, the N_{tries} was varied between 2 and 16. This parameter was mostly related to reliability, but enlarging its value worsened latency and power consumption (even though they were affected to a lower extent). For N_{slot}, we used the default value, 101. The results are reported in Table 6.4. The probability $1 - e^{pkt}$ that a packet was successfully delivered to its destination (reliability) quickly approached 1 when $N_{tries} = 2$, $1 - e^{pkt} = 0.98154$, while it was as high as 0.999999999999993 (14-nines) when N_{tries} was increased to 16. As expected, we managed to measure the number of lost packets different from 0 by only setting $N_{tries} = 2$. In this case, the measured packet loss ratio was $P_{lost} = 0.017$, which was close to the $1 - e^{pkt}$ estimate.

Table 6.4: Experimental results about the influence of N_{tries} on latency, reliability, and power consumption (measured on real devices).

	Latency								Reliability			Power Consumption				
N_{tries}	d_{min}	μ_d	σ_d	d_{p99}	d_{max}	$\hat{n}_{tra}$	$\hat{\mu}_d$	Max_d	P_{lost}	ϵ	$1 - \epsilon_{pkt}$	f_{tra}	f_{listen}	P	$\hat{f}_{tra}$	$\hat{f}_{listen}$
			[s]			[#]	[s]	[s]				$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$	μW	$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$
2	0.496	1.851	1.015	4.441	5.377	2.17	1.861	8.080	0.017	0.0963	0.98154	1.82	9.71	144.1	1.82	9.71
4	0.342	1.853	1.272	6.066	6.090	2.24	1.850	16.160	0.0	0.1102	0.99971	1.88	9.71	144.3	1.87	9.71
6	0.387	2.031	1.323	6.906	7.447	2.32	2.048	24.240	0.0	0.1388	0.99999	1.93	9.70	144.5	1.93	9.70
8	0.726	2.320	1.558	8.255	9.890	2.27	2.285	32.320	0.0	0.1197	7-nines	1.92	9.70	144.5	1.89	9.71
16	0.352	2.046	1.588	8.764	10.457	2.28	1.936	64.640	0.0	0.1244	14-nines	1.95	9.70	144.6	1.90	9.71

Increasing the retry limit had limited latency and power consumption effects, because the frame error probability ϵ in our setup was not particularly high (about $10 - 15\%$). This implies that the probability of performing many repetitions in a row was small, as demonstrated by the fact that $1 - e^{pkt}$ quickly converges to 1. As shown in the table, all statistics that refer to latency (including μ_d, σ_d, and

d_{p99}) were negatively affected by an increase in the retry limit. There was a good match between the mean value μ_d of the measured latency and its expected value $\hat{\mu}_d$. Also, the measured maximum latency d_{max} increased from 5.377 s to 10.457 s; however, this number is not statistically reliable because of the limited number of samples. In fact, the worst-case latency Max_d ranges, in theory, from 8.080 s when $N_{tries} = 2$ to 64.640 s when $N_{tries} = 16$.

The effects of N_{tries} on power consumption were mostly negligible. In particular, P was between 144.1 μW and 144.6 μW. This was for two reasons: first, the parameters selected for the network and application consumption depend mainly on idle listening; and second, multiple retries are performed only when frame transmission repeatedly fails, which was unlikely in our setup.

6.4 Practical application contexts

The results provided by the network model were comparable with measurements obtained from real devices. Hence, the model can be profitably used alone (i.e., without a real setup) to better investigate the effect of parameters N_{slot} and N_{tries} on performance indicators in specific operating conditions.

6.4.1 Leveraging the mathematical model

The model was used to analyze the behavior of TSCH in the case of frame error probability ϵ being increased to 0.4, denoting a hostile environment. The values of d_{min} depend on the configuration of the TSCH matrix; however, for our analysis, it was relevant, because it caused only a shift on the time axis (latency offset). For this reason, $d_{min} = 500$ ms were set for all plots shown in the figure.

Initially, the number of slots in a slotframe was varied between 11 and 201. The results are reported in Plot 1 of Fig. 6.4. To calculate P, we used (6.7), where the value $\hat{f}_{tra}$ was obtained from (6.17). As expected, the average latency, obtained from (6.18), increased linearly with N_{slot}, while power consumption decreased according to an inverse proportionality law: when N_{slot} was greater than ~ 58, power consumption fell below 250 μW. At the same time, the average latency exceeded 2.6 s. Decreasing N_{slot} well below such a threshold allowed a consistent reduction of latency; however, due to the increase in power consumption, the motes should be connected to an external power supply.

Plots 2 and 3 of Fig. 6.4 were obtained by changing N_{tries} from 1 to 16. As highlighted in both graphs, reliability $1 - \epsilon_{pkt}$, given by (6.1), increased as N_{tries} increased. A similar trend but with some relevant differences can be observed for the average latency μ_d in Plot 2 and the power consumption P in Plot 3. In both cases, the shape of the plot is seemingly the same, but in terms of the magnitude of the increase, the ranges in which values vary were different; the average latency

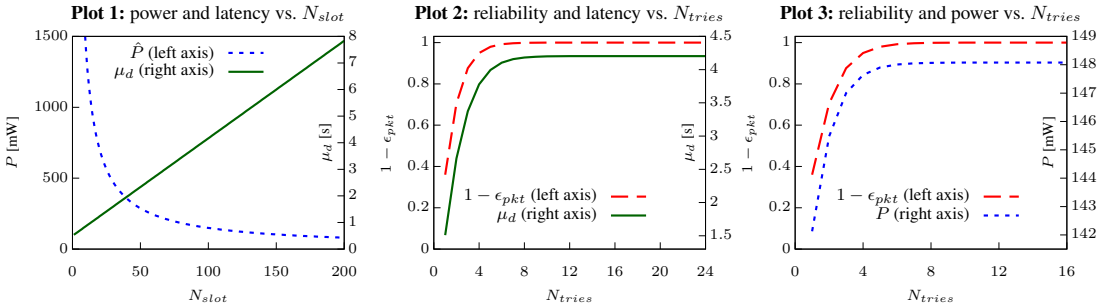


Figure 6.4: Influence of N_{slot} and N_{tries} on reliability, power consumption, and latency, evaluated using the proposed network model ($\epsilon = 0.4$, $N_{tries} = 16$ for Plot 1, $N_{slot} = 101$ for Plot 2 and Plot 3).

was influenced more, and increased from 1.5 s when $N_{tries} = 2$ to values slightly above 4 s when $N_{tries} \geq 5$.

In the same conditions, the power consumption increased from 142 mW to 148 mW, which was a modest increase. This is because most energy is spent on idle listening and initial transmission attempts (including the first retries possibly performed for each frame). In contrast, only a limited contribution is related to the additional retransmissions made available by increasing the retry limit since a small fraction of frames experienced many repeated failures.

In summary, increasing N_{slot} had a positive effect on power consumption, a negative effect on latency, and no effects on reliability. Alternatively, increasing N_{tries} had a positive impact on reliability, a negative effect on latency, and a slightly negative effect on power consumption (practically negligible for typical values of the frame error probability). Some working points of interest were identified from previous plots and analyzed using an extensive experimental campaign on real hardware.

6.4.2 Evaluation of relevant configurations

As stated in the introduction, WSNs are exploited in many application contexts with different and often conflicting requirements. This chapter identifies four sample configurations for a TSCH network, each of which suits the needs of some specific context. They are denoted as *default*, *high reliability*, *low latency*, and *low power consumption*, and are characterized by specific values for N_{slot} and N_{tries}.

The *default* configuration refers to out-of-the-box OpenWSN devices. To find suitable values for the other configurations' network parameters, experimental analysis was started from the plots in Fig. 6.4 and re-evaluated for $\epsilon = 0.13$, which more closely resembles the frame error probability typically observed in our experimental environment. The results are reported in Fig. 6.5. Then, three reasonable working points were sought on the plots, aimed at optimizing the performance indicators

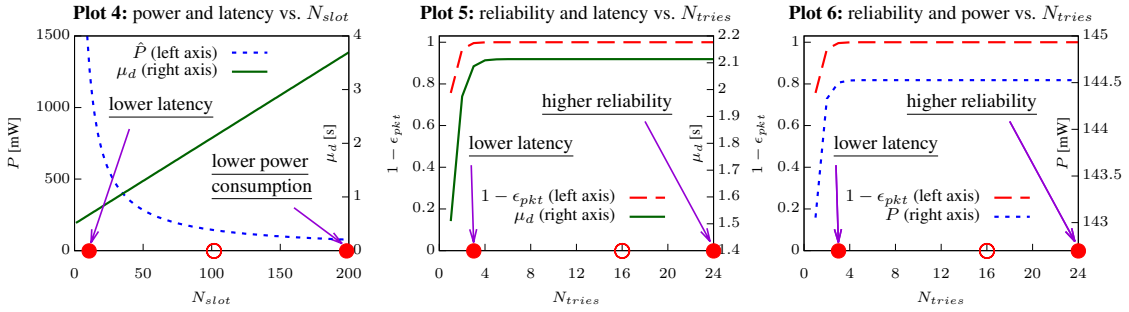


Figure 6.5: Influence of N_{slot} and N_{tries} on reliability, power consumption, and latency, evaluated using the proposed network model ($\epsilon = 0.13$, $N_{tries} = 16$ for Plot 4, $N_{slot} = 101$ for Plot 5 and Plot 6). Effects of moving working points—marked with solid red circles (•)—away from the *default* configuration—marked with empty red circles (∘)—are suitably labeled.

without excessively penalizing the others.

The effect of increasing or decreasing each parameter (either N_{slot} or N_{tries} changed in every plot) is shown in Fig. 6.5 using underlined labels. The working points in the figure and configurations are different. For instance, the working point in Plot 4 labeled "lower latency," achieved by reducing N_{slot} from 101 to 11, does not correspond to the *low latency* configuration. In fact, in the latter case, the value of N_{tries} was further decreased from 16 to 3, to lower the maximum latency besides equation (6.14). As will be shown, there is no optimal setting for N_{slot} and N_{tries} that suits all application contexts because improving a performance indicator unavoidably worsens at least one of the others.

In detail, the four configurations we considered are:

- *Default* ($N_{slot} = 101$, $N_{tries} = 16$): This configuration draws the default out-of-the-box setting of a WSN when motes are based on the OpenWSN OS, and develops a balanced trade-off among power consumption, reliability, and latency.

- *High reliability* ($N_{slot} = 101$, $N_{tries} = 24$): This configuration is characterized by a reliability level higher than the default case. In particular, 8 additional retries were granted to every frame transmission (N_{tries} was increased from 16 to 24).

- *Low latency* ($N_{slot} = 11$, $N_{tries} = 3$): This configuration is targeted for applications demanding low latency. In this case, N_{slot} was reduced by one order of magnitude (from 101 to 11), which lowered all statistical indices about latency (including the average and maximum values), and N_{tries} was set to 3 to provide a stricter limiting on the worst-case latency.

- *Low power consumption* ($N_{slot} = 201$, $N_{tries} = 16$): This configuration is conceived for applications in which batteries on motes must be replaced as seldom as possible. For this purpose, the value of N_{slot} was doubled from 101 to 201.

The duration of each experiment was set to 24 hours ($N_{sam} = 720$), and the purposely injected interfering traffic was the same as in the previous experiments. The results are reported in Table 6.5.

Table 6.5: Latency, reliability, and power consumption, measured on real devices, related to four configurations (characterized by different values of N_{slot} and N_{tries}) targeted to different application contexts.

Configuration			Latency						Reliability			Power Consumption		
Condition	N_{slot}	N_{tries}	d_{min}	μ_d	σ_d	d_{p99}	d_{max}	Max_d	P_{lost}	ϵ	$1-\epsilon_{pkt}$	f_{tra}	f_{listen}	P
			[s]					[s]				$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$	μW
Default	101	16	0.528	2.115	1.310	6.579	11.049	64.640	0.0	0.125	14-nines	1.91	9.71	144.4
High Reliability	101	24	1.470	3.090	1.320	7.450	9.360	96.960	0.0	0.132	20-nines	1.92	9.71	144.5
Low Latency	11	3	0.159	0.336	0.135	0.780	1.023	1.320	0.0042	0.142	0.9942	1.92	90.71	1262.4
Low Power Cons.	201	16	2.565	5.535	2.461	13.637	22.366	128.640	0.0	0.112	14-nines	1.92	4.78	76.5
Default (15-days)	101	16	0.522	2.114	1.289	6.393	12.382	64.640	0.0	0.126	14-nines	1.90	9.71	144.5

The *high reliability* configuration (second row in the table) is characterized by an extremely low packet loss probability. Under the hypothesis that frame attempts can be modeled as Bernoulli trials, reliability $1 - e^{pkt}$ is, in theory, as high as 0.99999999999999999983, or 20-nines. Although this parameter setting means a tangible increase of the theoretical worst-case latency Max_d concerning the default case, both the measured latency and the power consumption were only marginally affected, as the transmission error ϵ on the channel was not extremely high.

Concerning the *low latency* configuration, all statistics about latency were noticeably lower than the default case. In particular, the measured maximum ($d_{max} = 1.023\,\text{s}$) was very small and close to the theoretical worst-case value ($\text{Max}_d = 1.320\,\text{s}$), while the average time taken to perform a request/response exchange was $\mu_d = 0.336\,\text{s}$. Unfortunately, low latency settings led to a significant increase in the measured packet losses ($P_{lost} = 0.0042 = 0.42\,\%$), and a decrease of the estimated reliability ($1 - e^{pkt} = 0.9942 = 99.42\,\%$). As expected, the power consumption was about 9 times higher than the default case, which makes this solution mostly impractical for battery-powered devices.

Eventually, in the *low power consumption* configuration, the power consumed by motes fell to $76.5\,\mu\text{W}$, which was about half that in the default case ($144.4\,\mu\text{W}$). This was because, receiving motes, it must switch their radio transceivers on (to listen to the channel) less frequently. In particular, since the actual rate of packet

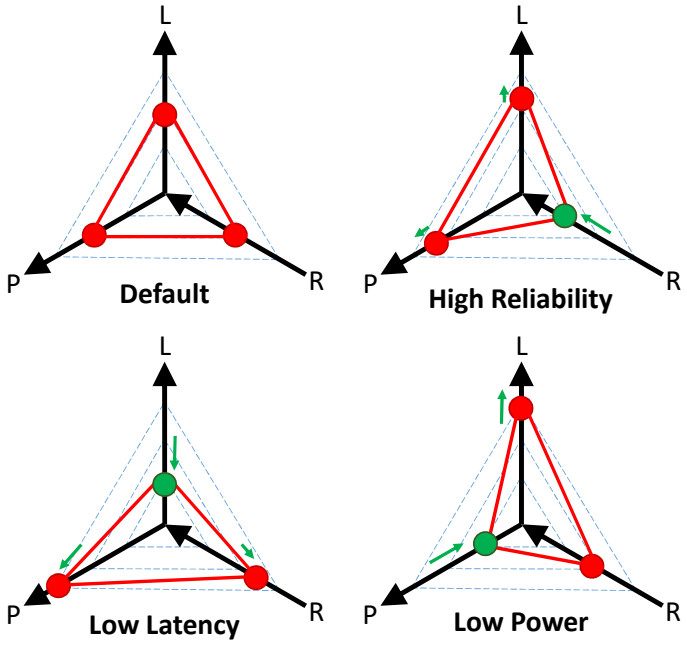


Figure 6.6: Effects of different parameter configurations (targeted to specific application contexts) on power consumption (P), reliability (R), and latency (L).

transmissions does not vary (depending on the sending period T_{app} of the application), f_{listen} reduced from 9.71 to 4.78 occurrences of idle listening per second. The main drawback of this setting is that there is a sharp increase in latency.

Fast characterization of the above configurations based on power consumption, reliability, and latency is described in the radar charts in Fig. 6.6, which highlight the related application contexts. Green points depict the quantity that mainly benefits from a given setting, while red points are the impacts (almost always adverse) that such settings imply on the other two doses. As demonstrated, optimizing all performance indicators at the same time is not possible.

The default configuration performance evaluation was left running after the first 24 hours to obtain an experimental sample to check the statistics' reliability. The experiment continued for 15 days. By doing so, $N_{sam} = 10800$ samples were collected (including those used to calculate the values in the first row of Table 6.5). The results are reported in the last row of the table. Further, in this case, no packets were lost, not even after two weeks of continuous operation, as evidenced by all `ping` requests succeeding. Measured latency was bounded to a maximum of 12.382 s, and the 99-percentile was 6.393 s. The results for this extended data set show a good match with those obtained over a single day, which confirms that the statistics of the latter are reliable.

6.5 Conclusions

In this chapter, a mathematical model of a TSCH-based network has been proposed, which provides three performance indicators relevant for WSNs: reliability, power consumption, and latency. This model relies on a few parameters that can be easily estimated from an actual setup deployed in the area of interest. These parameters can then be used in a later stage to evaluate the expected network performance when some TSCH configuration parameters are varied. To provide a realistic estimation of power consumption, a characterization of the latest version of OpenMote B motes was analyzed based on measurements performed on an actual setup. The related results have been presented here.

An analysis of three relevant application contexts, characterized by specific network configuration parameters (slotframe duration and retry limit), highlighted that reliability, power consumption, and latency are intimately connected. Therefore, it is impossible to optimize all the performance indicators at once, but their joint optimization must necessarily follow a holistic approach. TSCH is an exceptionally flexible solution, as the performance demanded by a wide range of application contexts can be easily achieved by suitably tuning these parameters. As an example, the analysis showed that for two-way communication in which every packet performs two hops overall (both directions considered), in typical operating conditions, it is possible to either decrease the average latency below $\sim \frac{1}{3}$ s or ensure 20-nines reliability, or, finally, reduce the power consumption of the network component by half relative to the default configuration.

Chapter 7

TSCH Predictor

A TSCH mathematical model was developed to predict the behavior of the 6TiSCH WSN. The model was proposed to analyze performance indicators, such as latency, reliability, and power consumption, for single-hop and multi-hop WSNs. The model also provides an estimation for P_{lost} packet loss ratio and ϵ_{pkt} packet loss probability. However, there was a limitation when the packet queuing was present in the nodes, this issue impacts during the performing experiments, and as a consequence, the latency on which the proposed model is built can not be valid anymore. Thus, the proposed model in Chapter 5 was ineffective at performing the WSN analysis. We propose an instrument to overcome the experimental analysis and estimate performance indicators without packet queuing limitation in the hardware node.

The 6TiSCH matrix directly influences the performance indicators, which means that we could modify the performance analysis by changing the parameters in the TSCH matrix. However, this modification is complex on real devices, such as OpenMote B devices, as the modification needs to be applied to the source code, then load the compiled code to the devices. This task is not very simple, as it takes time to modify the code and much other activity to perform an experimental evaluation. Therefore, we propose a tool to facilitate the deployment phases in industrial WSNs.

Simulation tools are the most-used tools in the engineering domain, which facilitates testing and deployment in the industry. WSN technologies have many simulating tools for wireless networks, such as NS2, OMNeT++, Prowler, OPNET, and TOSSIM. These simulators are powerful software platforms for simulating different protocols or networks, expecting strong knowledge to develop a tailor-made simulator for the TSCH WSN. However, many simulators are developed for 6TiSCH WSN, such as OpenWSN [151], Cooja for Contiki and Contiki-NG [152], and TSCH-Sim [153]. As far as we used, most simulators are complex for a normal user and made for protocol analysis. Thus, we propose a simulator that can easily simulate and predict a network's performances. The core of the *TSCH predictor* is developed

with a simple architecture that provides a framework to implement a new feature on top of the main core. The TSCH predictor is a tool to simulate and predict the performance indicators in the TSCH WSN. It provides a good prediction of power-consumption for the 6TiSCH running on the OpenMote B device.

7.1 Introduction

The WSN performance indicator suffers from Wi-Fi traffic. Therefore, the network simulator requires a realistic network simulation and considers network parameters and Wi-Fi disturbances to estimate the WSN network better. The proposed *TSCH predictor* is implemented by a simple architecture and includes essential factors, making its result trustable and comparable with real-world data. It can also adopt new features as needed.

The TSCH predictor was developed by considering OpenMote B hardware characteristic with OpenWSN OS on top of it. The TSCH predictor does not attach to the hardware, but it needs the energy measurement to calculate power consumption accurately. We compared the result of the *TSCH predictor* with the experimental data obtained from the testbed explained in Chapter 6. The comparison verifies that the *TSCH predictor* estimates the performance indicators with high accuracy. In this chapter, we compared the *TSCH predictor* with the experiments performed by the OpenMote B devices, and the OpenWSN OS was utilized for communicating via the 6TiSCH protocol.

The TSCH predictor has three main advantages: first, it stimulates the 6TiSCH protocol with different parameters, such as slotframe, retransmission, and other TSCH matrix settings; second, it overcomes the packet queuing issues and simulates the WSN with higher frequency , which it helps to have the timeslot less than 20 ms for each cell in every slotframe, without limitation; and third, it performs network simulation in parallel on different WSNs, making it more efficient and flexible. Further, the *TSCH predictor* is simple and has a user-friendly interface that makes it easy to configure and start simulations; it just requires the packet delivery probability and the TSCH matrix parameters to start the simulation.

In the following, the system architecture and implementation of *TSCH predictor* are described in Section 7.2, the simulation logic and interface details are presented in Sections 7.3 and 7.4, and details about the comparison between the simulation and the experimental results are presented in Section 7.5.

7.2 System Architecture

The *TSCH predictor* is based on Python. The core of the *predictor* is developed by a simple structure and implemented with the SimPy python library[1], which is a process-based discrete-event simulation framework based on Python.

The *TSCH predictor* architecture is defined based on the hardware and software structure. The proposed architecture is presented in Fig. 7.1, which shows that the system architecture is developed in three main layers: the **configuration layer**; the **system core**, which defines the hardware node, the 6TiSCH network, and the packet core; and the WSN **simulation core**, or the event generator for each slotframe. The simulation core generates events to perform simulations in the TSCH predictor. The data analysis is computed by utilizing the mathematical model proposed in Chapter 6, and it provides a performance analysis right after the network simulation. This layer provides a terminal interface and a Web API to facilitate the configuration of the simulation tool. The terminal interface of the TSCH predictor is developed for parallelization purposes, which helps simulate thousands of parallel WSNs with different configurations simultaneously.

All configuration and network parameters are defined in the configuration layer. This component is defined as a dynamic object while the simulator is running. All hardware and software parameters are defined here, as well as the delivery probability network parameter.

Figure 7.1 presents three subsections of the physical system: the **hardware node**, the **TSCH network**, and the **packet core**. The **hardware node** provides the hardware characteristics, such as the energy-consumption model, which helps to calculate power consumption for each node in the network. It also tracks all the

[1]https://simpy.readthedocs.io/en/latest/

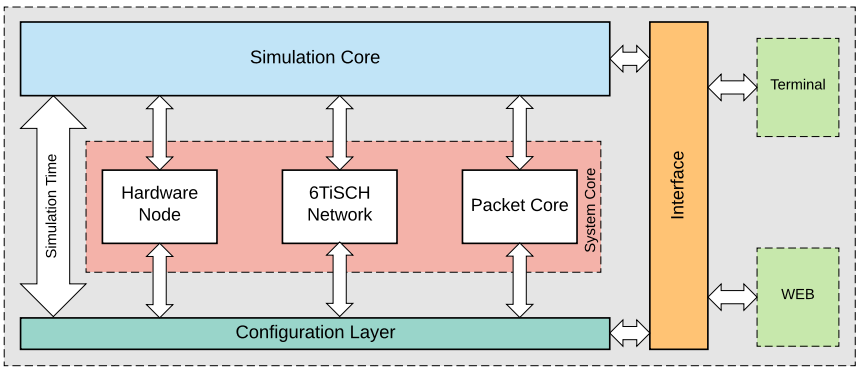


Figure 7.1: TSCH *predictor* Software Architecture.

packets transmitted in the network. The **TSCH network** represents the 6TiSCH matrix parameters and software configuration. The WSN is generated based on **TSCH network** by utilizing the **hardware node**. The **packet core** generates and controls the packet generation and controls the packet transmission between the nodes.

Moreover, it decides if the packet is successfully transferred to the next hop or not. The **TSCH network** provides a packet destination embedded into the WSN after the network topology is created. The probabilistic model of the collisions and Wi-Fi traffics are considered here to provide a truthful prediction of the WSN performance indicator. All three subsections are defined in different class objects to easily modify the simulator core and implement new features in the TSCH predictor.

The **simulator core** runs on top of the **system core**. The network topology is created here based on the TSCH network matrix, defined by the system core. The **simulator core** uses SimPy to develop a process-based discrete-event simulation. The simulator core defines each cell's events in the slotframe in the TSCH matrix to perform the simulation and, like a real device, sends a packet to the next hop in each slotframe in the cell. Each node sends a packet in their scheduled time, defined in their cell in the TSCH matrix. Each cell knows where to send data. After the network topology is created, the simulator core starts to send packets at a specific time defined in the parameters. The simulation core tracks latency, duplication, and delivery time and stores the results in the log file after finishing the simulation. After finishing the simulation, the analysis tool estimates the performance indicators by utilizing the formula proposed in Chapter 6.

7.2.1 TSCH predictor configuration layer

The TSCH predictor configurations are defined here as a global object in the TSCH predictor architecture, and it is accessible in the whole program. This layer is divided into three subsections: general configuration, flow configuration, and 6TiSCH matrix configuration.

General configuration

The simulator parameters are defined in the *general configuration* file, defined as a `JSON` format. The common network parameters and *simulation parameters* are defined, such as total simulation time, energy model, number of slotframes, and maximum number of attempts for transmission for each packet.

Flow configuration

This configuration defines the RPL and scheduler parameters for the network. It represents the source, destination, routing of the network, type of the scheduler,

and period of request/response for each packet. These parameters are used to build the cell and node instances in the node and TSCH matrix in the *system core*. The routing considers the tree and starts the network topology, which can change and simulate a new network topology with new parameters. The configuration file format is `JSON`, and it is defined as a global variable.

6TiSCH configuration

The 6TiSCH parameters are defined in the configuration file, with a `plain text` format. The parameters (e.g., slot offset, channel offset, source (scr), destination (dest), frame delivery probability [FDP: $1 - \epsilon_{frame}$] and ACK delivery probability [ADP: $1-\epsilon_{ack}$]) are defined in the file for each direction. Listing 7.1 shows the TSCH matrix format template, which defines the TSCH configuration network based on the IEEE802.15.4 standard. In this template, the first component in the row defines the cell slot offset or the slot number that its node will send the packet to. This means that if this value is 18, the node will be scheduled $18 * 20 = 380ms$ to transmit data for each iteration in the slotframe period. The second component in the row defines the channel offset. The third value is the source ID, which starts at 0 and it ends at the total number of nodes. The next parameter is the destination ID, which indicates the packet destination or the destination node ID. If the source is 0 and the destination is 1, the packet will be sent from node 0 to node 1 ($0 \rightarrow 1$).

The most crucial TSCH predictor parameter is the simulator's fundamental parameter, the frame delivery probability. The 5^{th} and 6^{th} parameters are the frame delivery probability and the ACK delivery probability, respectively. These values are usually defined as inputs for the simulator, and they are computed from the `PING` value utilities.

Listing 7.1: TSCH Matrix definition

```
# offset channel_offset src dest FDP ADP
1 0 0 1 0.887411967465 0.887411967465
18 1 1 2 0.887411967465 0.887411967465
100 1 2 3 0.887411967465 0.887411967465
```

7.2.2 System core

The core is divided into *hardware node*, *6TiSCH network*, and *packet core*.

Hardware node

The *hardware node* in the WSN architecture is developed in this layer to define the hardware characteristics of the OpenMote B , which can adopt other hardware characteristics too. The *hardware node* defines all the node functionalities

Table 7.1: Energy consumption for different types of actions within a slotframe matrix with OpenMote B motes. In **bold** quantities used in the *hardware node*.

Quantity	Action(s)	Slot offset	Energy [μJ]	Size [bytes]
$E^f_{rx\_data}$	RX DATA frame	16	178	87
$E^f_{tx\_ack}$	TX ACK frame	16	106	33
$E^f_{tx\_data}$	TX DATA frame	98	187	90
$E^f_{rx\_ack}$	RX ACK frame	98	79	33
E^f_{listen}	Idle listening	16	**138**	-
E^f_{comp}	Computation	-	628	-
E^f_{rx}	RX DATA + TX ACK	16	**284**	120
E^f_{tx}	TX DATA + RX ACK	98	**266**	123

(e.g., `rxDATA`, `txDATA_waitACK`, `rxDATA_txACK`, `txDATA_rxACK`, `listen`, `queue`, and `dequeue`). These functionalities help send and receive requests from each node and count the total number of sending and receiving packets. The total energy consumption of each node is also calculated in this layer. The total energy consumption is calculated based on the model presented in Chapter 6. The energy consumption value of each slotframe is reported in Table 7.1. The quantities are used to obtain the total power consumption in each node.

The 6TiSCH network

The *6TiSCH network* is responsible for importing the TSCH matrix configuration and global configuration for the WSN. It then creates cells for each node based on the parameters defined in the configuration files. Nodes are generated from the node object and are used inside the `tsch_event` function. The node has all the functionality to send and receive packets. The frame delivery probability is used here to exchange the frame in each cell.

Packet core

The packet core is implemented to generate a packet for the simulation layer. This layer generates the packets for each node in each event; then, the cells are responsible for sending the packet from source to destination in their scheduled time slot. The node tracks delivered or lost packets in each request. The packet object defined in the *packet core* has many methods that help track the frames, the most used of which are `get_next_node`, `arrived_next_hop`, `not_arrived_next_hop`,

arrived, `is_lost`, and `is_dup_packet`.

7.2.3 Simulation core

The **simulation core** generates events to run the simulation. The TSCH predictor is based on the **SimPy** module that provides a framework to implement a discrete event simulator. This layer orchestrates all processes and layers.The **simulation core** first imports the matrix and node configurations from the *configuration layer*. Then it creates the SimPy environment instance that helps generate event simulation. This step invokes the `TSCH network` object to create an event for each cell in the TSCH matrix by assigning them to the `hardware node`. The SimPy environment schedules all nodes as an event. All the node objects are created at this stage, and they are attached to the simulator core. In this step, the packet core invokes and generates a packet for each flow defined in the *flow configuration*. The events generated in the *simulation core* are responsible for generating a packet for each slotframe for the nodes. The events are attached to the SimPy environment as a process. The simulation result appears after the total time of simulation finishes in the SimPy environment.

7.3 Simulation Logic

The simulation of the delivery of frames and acknowledges is performed by comparing a random number (with uniform probability distribution) with the corresponding delivery probability value. This means that the frame would not be succeeded if the frame delivery probability is more than the probability of the air (system). Further, if the packet does not have the chance to be sent, it will retry in the next slotframe until the total number of tries becomes overflow. The retry parameter (*TXT_RETRIES*) is defined in the general configuration file. Nonetheless, the frame is lost if the frame cannot satisfy the condition for all retry attempts.

The simulation happens in the *TSCH event* function of the simulation core. As shown in the logic, two conditions are defined for the request (send frame) and response (ACK) frames to check if the packet arrives at the node. When the new packet is generated in each event, the event generates a uniform random number x ($0 \leq x \leq 1$) and checks if the frame delivery probability FDP ($FDP = 1 - \epsilon$) is more than x, at which point this condition decides that the frame has reached the destination. The same condition is used for ACK frames in the second step, and if both conditions are satisfied, the packet reaches the destination node. The simulation logic is presented in Algorithm 1.

Algorithm 1: Simulation logic at the *TSCH event*

```
FDP ← (1 − ϵ)
ACK ← (1 − ϵ_ACK)
while TRUE do
  if random.uniform(0,1) ≤ FDP then
    DATA frame arrived in subsequent node
    if random.uniform(0,1) ≤ ACK then
      ACK frame arrived to source node
    else
      ACK frame did not arrive at source node
    end if
  else
    ACK frame sent but did not arrive at the following node
  end if
end while
```

7.4 Interfaces

The TSCH predictor has two interfaces, terminal, and Web. The terminal interface runs on the single and multi-core CPUs and clusters, and it can simulate multi-WSNs simultaneously on the same server.

The terminal interface requires a configuration file to run the predictor. The predictor performs the simulation and stores the experimental results in a file after finishing the simulation. An example is shown in Listing 7.2.

Listing 7.2: Running the predictor via Terminal

```
$ python TSCHpredictor.py EXAMPLE_101_16_0-0-0-0.conf
```

The TSCH predictor was developed to provide a simple and easy interface to perform the simulation. The parameter definition and configuration are user-friendly, as shown in Fig 7.2, the web interface is designed to be simple and effective in configuration. The interface takes the parameter values and then stores them in the TSCH predictor configuration files. It also provides a complete report after finishing the simulation and performance analysis based on the experimental data simulated by the software. This feature is available for both interfaces. The terminal interface provides the information by performing an analysis using the post-processing script after the simulation, and the Web interface provides an analysis by selecting this feature. The analysis provides the performance indicators in both interfaces. This feature was developed based on the model proposed in Chapters 5 and 6.

7.5 Results

Two sets of experiments were performed to analyze the effectiveness of the TSCH predictor: one performed on the actual device, and the other experiments performed on the simulator. The results were analyzed using the analyzer feature in the TSCH predictor to see how much the THSC predictor results were similar to the experiments archived from real devices.

$N_{slotframe}$ and N_{tries} were varied for each experiment. The round trips of the packets on the real devices were obtained using the `PING` utilities. Then, the frame delivery probability (ϵ) was obtained for each experiment from the round-trip logs. The same analysis was performed in Chapter 6 to evaluate the frame delivery probability. Table 7.3 shows some of the experiments performed in the previous chapter, where the results are presented to validate the proposed simulator in this chapter. The 24-hour experiments were performed for each configuration, and the results in the first section are taken from Table 6.5 in Chapter 6. A different set of configurations was selected in the TSCH matrix to obtain the results. The set configurations were [11,3] , [101,16], [101,24], and [201,16], where the first and second components in the setup were $N_{slotframe}$ and N_{tries}, respectively.

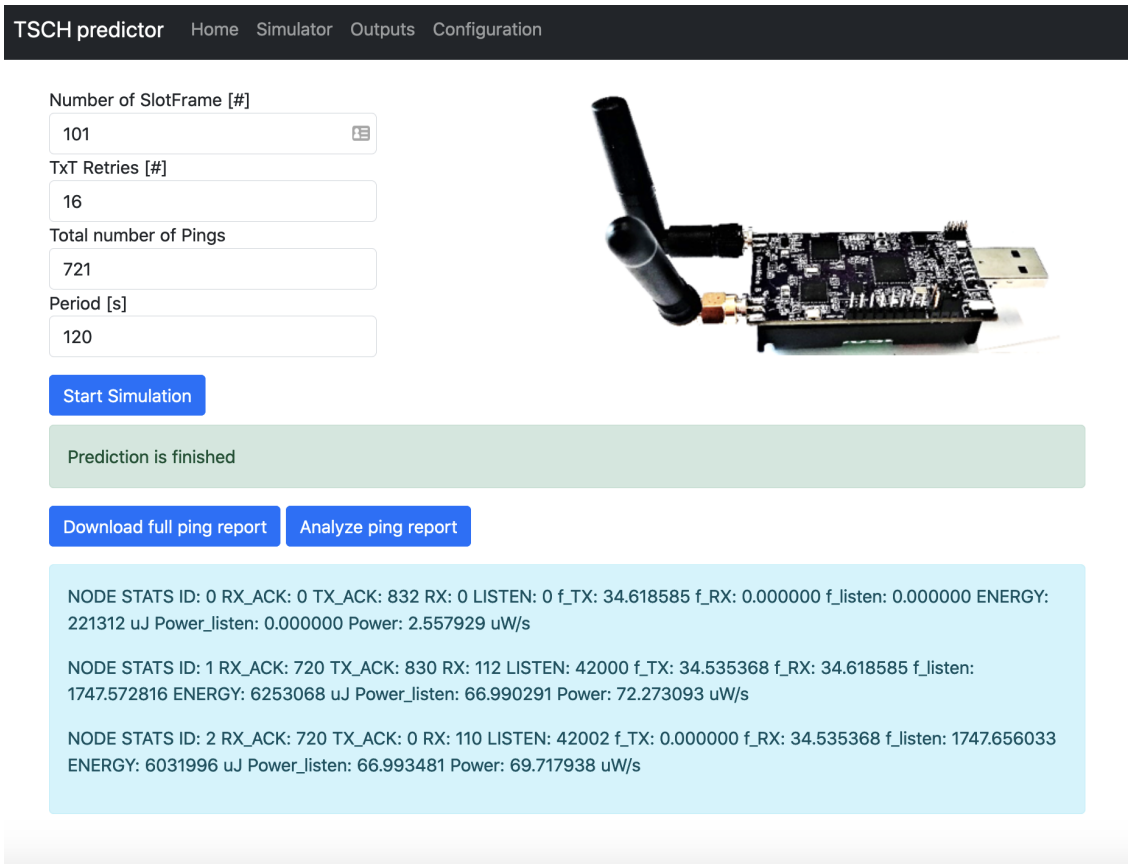


Table 7.2: TSCH predictor Web interface

Table 7.3 presents the analysis of the simulation versus experiments on real devices. By changing the configuration, the performance indicators changed. Reliability decreased when N_{tries} decreased in the WSN. When the $N_{slotframe}$ was set to high value, the latency increased. This is because latency is bounded in $d_{min} \leq d_i < d_{min} + T_{sframe}$, where d_i is the latency that packet i experiences, and if the requested packet is lucky enough, d_{min} becomes $(N_{slotframe} - N_{rx,slotframe} + N_{tx,slotframe}) * T_{slot}$. Thus, when the $N_{slotframe}$ value is high, latency increases. However, when the $N_{slotframe}$ increases, power consumption decreases. This is due to the *listen frequency* (f_{listen}) value. Therefore, if the $N_{slotframe}$ is higher, the f_{listen} decreases, as does the power consumption. Detailed explanations of these analyses are presented in Chapter 6.

The different values of ϵ are presented in the table, evaluated with the other experimental setup. The ϵ values were utilized for the TSCH predictor to simulate the nodes. These values were used to prepare the configuration file in the *TSCH matrix*. The slotframe offset, ϵ_{tx}, and ϵ_{ACK} are defined in the TSCH matrix.

The simulation was performed for different set configurations for one year long.

Then, the performance analyses are performed for each simulation. The experimental and simulation analyses are reported in Table7.3, which shows that the simulation values are good enough to predict the WSN performances.

In summary, The ϵ values calculated by the simulation were close to the ϵ values obtained from real devices. The ϵ values have 99.5%, 99.7%, 99.7%, and 99.8% similarity between the real and simulated values for $[11, 3]$, $[101, 16]$, $[101, 24]$, and $[201, 16]$, respectively. Further, the mean latency μ_d for both cases was compared, and both around 99% similar to each other, with a standard deviation of 94.5%–99.38%. The power consumption and the frequency of listening and transmission obtained from the TSCH predictor were also close (about 99.90% similarity) to the real devices. Note that the simulation was performed in a different configuration; it was cross-checked using the real round-trip data obtained by the `PING` utilities via OpenMote B devices.

Table 7.3: Simulation data compared with real experimental data obtained from OpenMote B

TSCH Conf.		Latency								Reliability			Power Consumption				
N_{slot}	N_{tries}	d_{min}	μ_d	σ_d	d_{p99}	d_{max}	$\hat{n}_{tra}$	$\hat{\mu}_d$	Max_d	P_{lost}	ϵ	$1-\epsilon_{pkt}$	f_{tra}	f_{listen}	P	$\hat{f}_{tra}$	$\hat{f}_{listen}$
		$[s]$					$[\#]$	$[s]$	$[s]$				$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$	$[\mu W]$	$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$
Real data with Openmote B																	
11	3	0.159	0.335	0.134	0.780	1.023	2.32	0.338	1.320	0.4166	0.1428	0.9941	1.93	90.7	1262.49	1.94	90.71
101	16	0.522	2.117	1.293	6.398	12.382	2.29	2.12	64.640	0	0.1263	1	1.91	9.71	144.494	1.91	9.71
101	24	1.47	3.089	1.32	7.45	9.36	2.31	3.10	96.96	0	0.1323	1	1.92	9.71	144.554	1.92	9.71
201	16	2.565	5.534	2.46	13.637	22.366	2.25	5.60	128.64	0	0.1125	1	1.92	4.78	76.5805	1.87	4.79
Simulation with TSCH predictor																	
11	3	0.160	0.329	0.142	0.780	1.240	2.31	0.339	1320	0.5761	0.1435	0.9940	1.93	90.7	1262.53	1.93	90.71
101	16	0.520	2.105	1.301	6.420	14.500	2.29	2.12	64.640	0	0.1266	1	1.91	9.71	144.496	1.91	9.71
101	24	1.46	3.078	1.334	7.440	18.400	2.31	3.09	96.96	0	0.1327	1	1.92	9.71	144.551	1.92	9.71
201	16	2.58	5.581	2.443	13.98	30.18	2.25	5.61	128.64	0	0.1127	1	1.88	4.79	76.3954	1.88	4.79

Our results show that the *TSCH predictor* provides a good simulation and estimation value for the performance indicators in different configurations. The simplicity of performing a simulation and analyze the performance indicators is an advantage in the TSCH predictor. Also, it could help optimize the network efficiency by choosing the suitable parameter in the TSCH matrix, which is $N_{slotframe}$ or $N_{retries}$ or both together. The TSCH simulator overcame the packet queuing issue in the WSN analysis. The TSCH predictor is designed to help engineers validate 6TiSCH configurations before deploying networks, and the predictor can give a perfect estimation for network performance indicators, especially power consumption. This analysis could help predict when to change battery-powered WSN nodes.

Chapter 8

Results

This chapter summarizes some of the important results from this thesis, proposes future work, and addresses existing issues raised by IoT and WSN in Industry 4.0. This thesis focused on two topics: standardization of the Internet of Things (IoT) platform and modeling and simulation in the 6TiSCH Wireless Sensor Network for Industry 4.0.

8.1 OPC-IoT

The OPC-IoT platform was proposed to overcome IoT standardization problems in Industry 4.0. The proposed architecture was based on OPC-UA, which complies with the IEC62541 standard. The platform was implemented in three layers–data collector, gateway, and data validation–and storage on the server-side. The platform was adopted with ProfiNet and Modbus protocols, which simplify shop-floor communication with industrial machines and exchange data with the OPC-UA protocol. The platform can store data on Cassandra and MongoDB. Kafka was also used for data broadcast systems between different parties in the system. This platform was evaluated against the DIIG gateway (see [32]), with the Kaa IoT platform as data storage in the architecture. Experimental analysis was performed to analyze the system's performance indicators (throughput, round trip, fairness, and scalability).

Experimental analysis was performed to evaluate both platforms' performance indicators and select the best data collector among them. The Cassandra database and the MongoDB were the same concerning throughput in the platforms; however, in the OPC-IoT platform, the Cassandra database was better when the total number of concurrent clients was increased. As shown in Fig. 8.1, the test was performed with the Cassandra database for 1,000 clients, and the total throughput was around 8,000 packets / second after increasing the number of clients.

As explained in Chapter 1, the OPC-IoT architecture is based on the OPC-UA protocol, which is suggested by RAMI 4.0 as a communication layer. This protocol provides a communication layer to the shop floor without any additional gateway or devices. Although, the gateway was proposed in the architecture for data conversion for a different protocol in the OPC-IoT platform. The proposed gateway communicates with industrial protocols, such as Modbus or Profinet, without adding new components or devices on the shop floor and communicating with OPC-UA–compatible devices. The performance evaluation was performed using Profinet, and the delivery acknowledgment is an advantage for the proposed architecture.

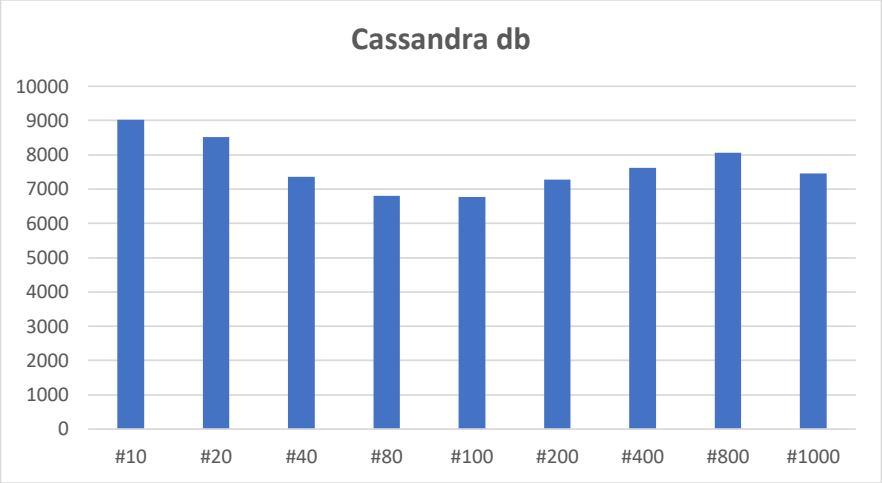


Figure 8.1: OPC-IoT: scalability test for 10 to 1000 clients

8.2 Fog Architecture

IFog4.0 is a fog architecture proposed to overcome the centralization problem described in Chapter 3. The architecture was proposed and implemented to comply with the RAMI 4.0 road map. The architecture was developed based on open-source components to make it more accessible for industry SMEs. The architecture was proposed with modular capability, with many preinstalled components, such as IDE, visualization, and ERP. Further, it was designed to be able to install customized features. Fog management was developed to control the fog platform easily.

To verify the proposed architecture's implementation and effectiveness, IFog4.0 was deployed in an industrial use case: a gas regulation station. The IFog4.0 was communicating with the real PLC, sending a decision command to the gas station

components, such as controlling the water bath heater's temperature and cleaning the dry gas filter in the gas station.

The IFog4.0 architecture was proposed for industrial applications to deliver a framework that could provide fast implementation of Industry 4.0 in SMEs. As shown in Chapter 3, *IFog4.0* provides a set of development components for enabling "Industry 4.0" based on RAMI 4.0. The **IFog4.0** platform introduced a scalable architecture based on open-source resources, such as *Docker* and *Node-RED*. Open source components make it possible to achieve greater accessibility. The *IDE* tool embedded in IFog4.0 was specifically tailored for this purpose. Finally, the architecture also enables component development by using the Docker SDK, deploying it on the platform using the fog-management tool. The proposed workflow guides the end-user to develop a custom application for their needs and install it on their system.

8.3 Single-hop WSNs

Experimental analysis in Chapter 1.2 shows that WSN communication quality suffers from background traffic, such as Wi-Fi and many other devices that work in the same frequency spectrum as WSNs. This issue negatively affects WSN performance indicators. To prevent this issue, a mathematical model was proposed in Chapter 5 to predict and estimate performance indicators in WSNs. The analysis was performed for a single-hop WSN (such as start topology), the most common WSNs topology. This topology is frequently utilized in home automation in the smart home industry.

Further, in Chapter 6, the model was extended to multi-hop networks, with the realistic power-consumption model and a latency and reliability model. Experimental analysis was performed to validate the model estimation. The OpenMote B device running the 6TiSCH protocol was employed to perform all experiments, where two configurations were applied–one with channel hopping enabled and another with channel hopping disabled. Both experiments were performed to analyze the effectiveness of channel hopping against Wi-Fi interference. The experiments were performed with controllable background traffic to analyze better the effect of the Wi-Fi interference on the WSNs.

The experimental analysis verified that the proposed model for a single-hop TSCH network estimates performance indicators with high similarity. A CDF analysis of channel hopping enabled and disabled (which was repeated from Chapter 5) was obtained to verify the latency similarity, as shown in Fig. 8.2 and Fig. 8.3. The CDF analysis verified that the proposed mathematical model could estimate the latency with high similarity when different background traffic was present. The $\mathcal{I}_{\emptyset}^{(+)}$, $\mathcal{I}_{6}^{(+)}$, and $\mathcal{I}_{6,6}^{(+)}$ define the number of background traffic in each experiment (zero, one, or two activated Wi-Fi interferences, respectively).

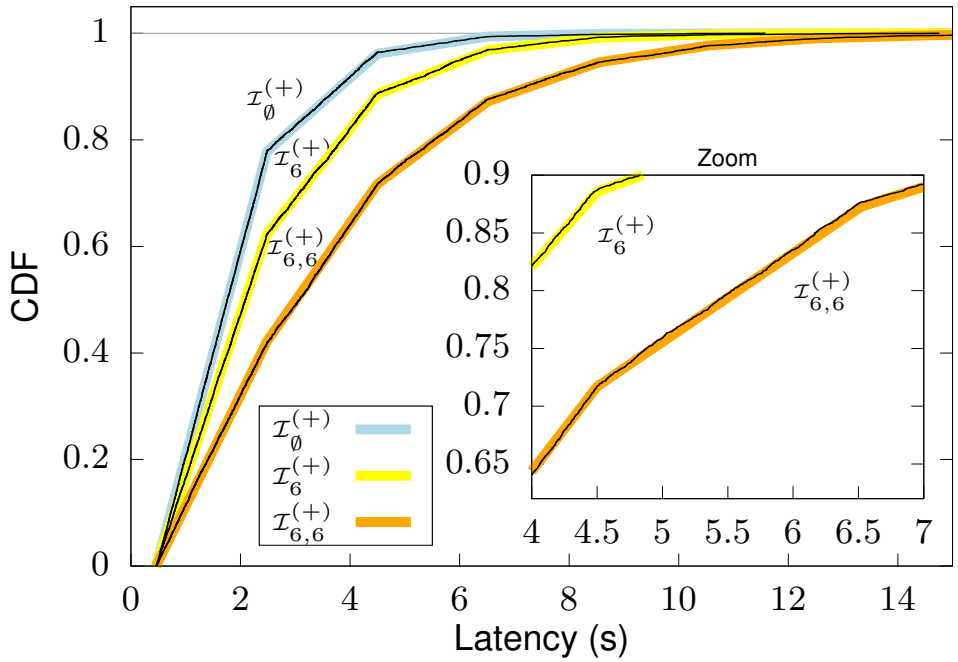


Figure 8.2: Measured and theoretical CDFs of d (channel hopping disabled).

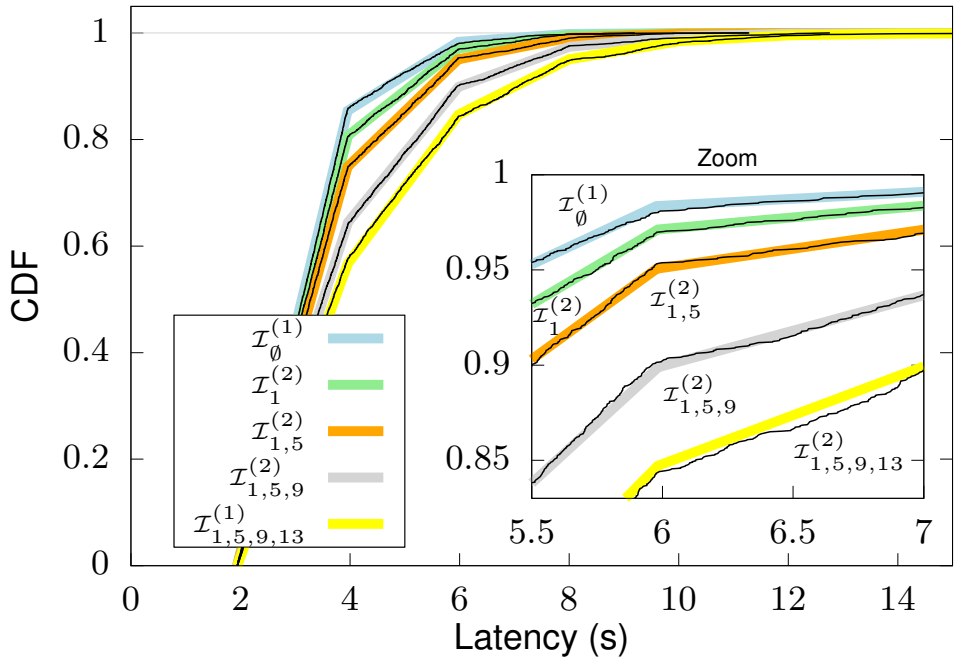


Figure 8.3: Measured and theoretical CDFs of d (channel hopping enabled).

This work's main result was not just estimating the network's performance indicator; the proposed models were developed to help engineers select the right configuration for 6TiSCH devices based on background traffic and noisy environments to satisfy the requirements. Fig.8.4 shows the main concept of this methodology. The model was developed to validate the 6TiSCH configuration with the requested requirements, and if it does not perform similarly, then the method must be repeated, and if it is verified, then the configuration may be finalized. This methodology and model were implemented for single-hop networks; the multi-hop model was proposed with a realistic power-consumption model and presented in Chapter 6 for the TSCH multi-hop network.

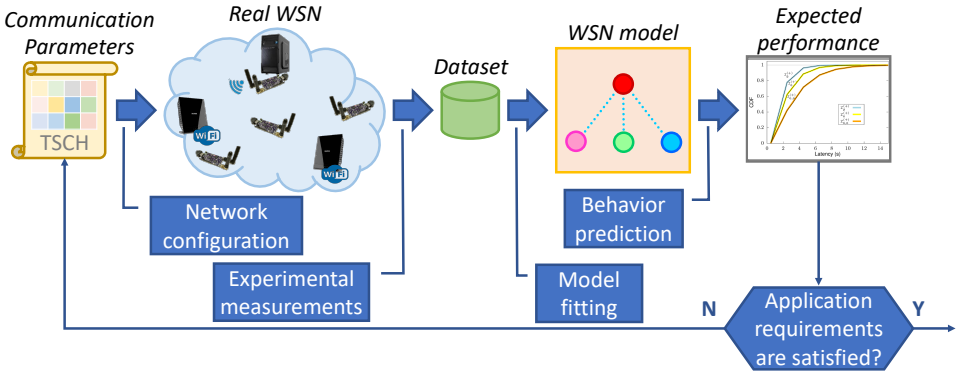


Figure 8.4: Main idea of the single-hop methodology

8.4 Multi-hop WSNs

The multi-hop mathematical model was proposed for the TSCH network in Chapter 6. The model was verified by the latency values stored by ping requests. The latencies were applied using the model as an input value, and the performance indicators were estimated with the proposed mathematical model.

Experiments were performed to evaluate the effect of different TSCH parameters on performance indicators, demonstrating that increasing the number of slots in the TSCH matrices increases latency. However, the transmission frequency decreases eventually; thus, power consumption decreases too. Many experiments were performed for different N_{slot} values, and the results are reported in Table 8.1 (the original observations are reported in the results section in Chapter 6). Further, other interesting experiments were performed to evaluate the number of transmissions on the performance indicators. Table 8.2 shows that if the number of transmission tries decreases, the power consumption and reliability also decrease.

The proposed model relies on a few parameters that can be easily estimated from an actual setup deployed in the area of interest. These parameters can then

Table 8.1: Experimental results about the influence of N_{slot} on latency, reliability, and power consumption (measured on real devices).

	Latency								Reliability			Power Consumption				
N_{slot}	d_{min}	μ_d	σ_d	d_{p99}	d_{max}	$\hat{n}_{tra}$	$\hat{\mu}_d$	Max_d	P_{lost}	ϵ	$1-\epsilon_{pkt}$	f_{tra}	f_{listen}	P	$\hat{f}_{tra}$	$\hat{f}_{listen}$
			[s]			[#]	[s]	[s]				$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$	μW	$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$
11	0.212	0.409	0.194	1.231	1.438	2.34	0.399	7.040	0.0	0.148	12-nines	2.00	90.70	1262.8	1.95	90.71
31	0.491	0.982	0.431	2.301	3.419	2.27	0.969	19.840	0.0	0.119	14-nines	1.91	32.06	453.0	1.89	32.06
51	0.258	1.024	0.649	3.007	3.054	2.25	1.021	32.640	0.0	0.110	15-nines	1.88	19.41	278.3	1.87	19.42
91	0.497	1.741	1.046	4.861	5.397	2.25	1.858	58.240	0.0	0.110	15-nines	1.87	10.80	159.4	1.87	10.80
101	0.352	2.046	1.588	8.764	10.457	2.28	1.936	64.640	0.0	0.124	14-nines	1.95	9.70	144.7	1.90	9.71
151	2.877	5.036	1.755	8.846	14.557	2.25	5.135	96.640	0.0	0.110	15-nines	1.85	6.44	99.0	1.87	6.43
201	0.726	4.216	2.880	12.131	14.050	2.36	4.193	128.640	0.0	0.153	12-nines	1.97	4.78	76.7	1.97	4.78

Table 8.2: Experimental results of the influence of N_{tries} on latency, reliability, and power consumption (measured on real devices).

	Latency								Reliability			Power Consumption				
N_{tries}	d_{min}	μ_d	σ_d	d_{p99}	d_{max}	$\hat{n}_{tra}$	$\hat{\mu}_d$	Max_d	P_{lost}	ϵ	$1-\epsilon_{pkt}$	f_{tra}	f_{listen}	P	$\hat{f}_{tra}$	$\hat{f}_{listen}$
			[s]			[#]	[s]	[s]				$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$	μW	$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$
2	0.496	1.851	1.015	4.441	5.377	2.17	1.861	8.080	0.017	0.0963	0.98154	1.82	9.71	144.1	1.82	9.71
4	0.342	1.853	1.272	6.066	6.090	2.24	1.850	16.160	0.0	0.1102	0.99971	1.88	9.71	144.3	1.87	9.71
6	0.387	2.031	1.323	6.906	7.447	2.32	2.048	24.240	0.0	0.1388	0.99999	1.93	9.70	144.5	1.93	9.70
8	0.726	2.320	1.558	8.255	9.890	2.27	2.285	32.320	0.0	0.1197	7-nines	1.92	9.70	144.5	1.89	9.71
16	0.352	2.046	1.588	8.764	10.457	2.28	1.936	64.640	0.0	0.1244	14-nines	1.95	9.70	144.6	1.90	9.71

be used later to evaluate the expected network performance when some TSCH configuration parameters vary.

An analysis of three relevant application contexts characterized by specific network configuration parameters (slotframe, duration, and retry limit) highlighted that reliability, power consumption, and latency are intimately connected. Therefore, it is impossible to optimize all performance indicators simultaneously; indeed, their joint optimization must follow a holistic approach (see Fig. 8.5). TSCH is a flexible solution, as the performance demanded by a wide range of application contexts can be easily achieved by tuning its parameters. As an example, we showed that, for two-way communication where every packet performs two hops overall (both directions considered), in typical operating conditions, it is possible to either decrease the average latency below $\sim \frac{1}{3}$ s or ensure 20-nines reliability, or, finally, reduce the power consumption of the network component by half concerning the default configuration. Note that the model is proposed if there is no packet awaiting

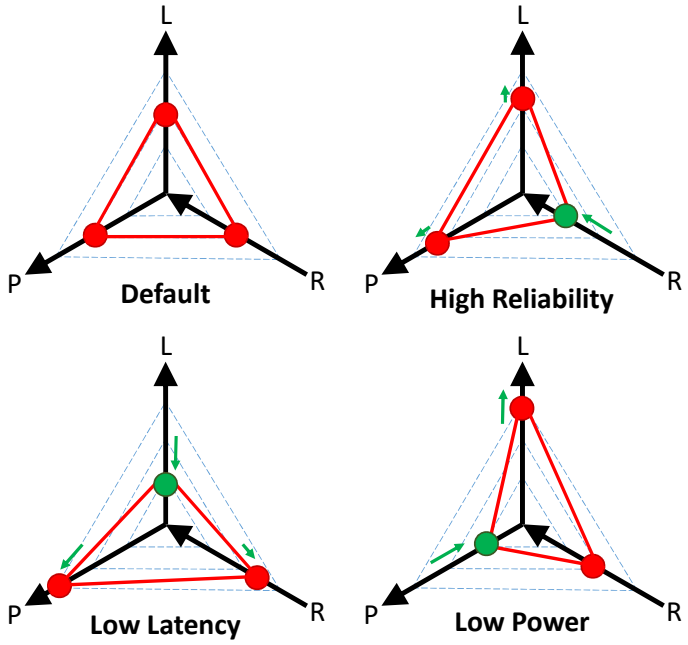


Figure 8.5: Effects of different parameter configurations (targeted to specific application contexts) on power consumption (P), reliability (R), and latency (L).

queuing of the OpenMote devices. This is not a limitation due to the frequency rate of the communication in many WSN scenarios. Nevertheless, the *TSCH predictor* was proposed to provide perfect estimations in the TSCH network.

8.5 TSCH predictor

The TSCH predictor was proposed to provide a simple and effective simulator for the TSCH network. This simulator was designed to be user-friendly and straightforward for any engineer with any background to simulate their design and predict the TSCH WSN configuration's performance indicators before deploying their design in the field. The predictor provides terminal and Web interfaces to run simulations and analyses.

The simulator result was compared with the experimental results performed in Chapter 6. The result shows that the TSCH predictor provides a realistic estimation compared to the testbed's real value. Table 8.3 reports the comparison between the *TSCH predictor* and experimental results performed on the real testbed.

Table 8.3: Simulation data compared with real ping data

TSCH Conf.		Latency								Reliability			Power Consumption				
N_{slot}	N_{tries}	d_{min}	μ_d	σ_d	d_{p99}	d_{max}	$\hat{n}_{tra}$	$\hat{\mu}_d$	Max_d	P_{lost}	ϵ	$1-\epsilon_{pkt}$	f_{tra}	f_{listen}	P	$\hat{f}_{tra}$	$\hat{f}_{listen}$
				$[s]$			$[\#]$	$[s]$	$[s]$				$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$	$[\mu W]$	$[\cdot 10^{-5}]$	$[\cdot 10^{-4}]$
Real data with Openmote b+																	
11	3	0.159	0.335	0.134	0.780	1.023	2.32	0.338	1.320	0.4166	0.1428	0.9941	1.93	90.7	1262.49	1.94	90.71
101	16	0.522	2.117	1.293	6.398	12.382	2.29	2.12	64.640	0	0.1263	1	1.91	9.71	144.494	1.91	9.71
101	24	1.47	3.089	1.32	7.45	9.36	2.31	3.10	96.96	0	0.1323	1	1.92	9.71	144.554	1.92	9.71
201	16	2.565	5.534	2.46	13.637	22.366	2.25	5.60	128.64	0	0.1125	1	1.92	4.78	76.5805	1.87	4.79
Simulation with TSCH predictor																	
11	3	0.160	0.329	0.142	0.780	1.240	2.31	0.339	1320	0.5761	0.1435	0.9940	1.93	9.07	1262.53	1.93	90.71
101	16	0.520	2.105	1.301	6.420	14.500	2.29	2.12	64.640	0	0.1266	1	1.91	9.71	144.496	1.91	9.71
101	24	1.46	3.078	1.334	7.440	18.400	2.31	3.09	96.96	0	0.1327	1	1.92	9.71	144.551	1.92	9.71
201	16	2.58	5.581	2.443	13.98	30.18	2.25	5.61	128.64	0	0.1127	1	1.88	4.79	76.3954	1.88	4.79

8.6 Conclusion

This thesis explored Industrial IoT middleware solutions and adopted them within the Industrial Protocol to be more compliant with the RAMI 4.0 road map for Industry 4.0. The OPC-IoT platform proposed here was developed to overcome standardization complexity, which is a property shown by any ordinary IoT platform. The OPC-IoT platform was first investigated by performing experimental analyses, then compared with a similar commercial IoT platform. The OPC-IoT platform overcame the compatibility issues arisen among the different IoT node vendors on the shop floor. Furthermore, since centralization is always challenging for an IoT platform in factories, the IFog4.0 system was also proposed to overcome that issue, shaping a decentralized architecture for IoT devices, by considering the RAMI 4.0 road map. The proposed architecture was developed and deployed for a gas regulation station's real scenario. That station was emulated with PLC devices, and IFog4.0 communicated with it and controlled the activity in the real scenario. IFog4.0 demonstrated that the new implementation could leverage the deployment of the IoT end-node in industry applications, and increase reliability and security by including a decentralized architecture in the whole Industrial IoT. Both proposed architectures, OPC-IoT and IFog4.0, have been introduced to overcome the standardization complexity of the IoT in Industry 4.0. Once developed and implemented, experimental analyses were performed to shape the performance indicators. The trials demonstrated that the proposed architecture could orchestrate the various standard included in the RAMI 4.0 road map. Hence, the proposed architecture may help researchers and SMEs adapt their factory communication and networks to the Industry 4.0 road map, so that a better and more reliable data acquisition can be achieved in Industry 4.0.

This work also focused on WSNs for industrial applications. As discussed in Chapter 1.2, WSNs suffer from background traffic caused by interfering WiFi and other wireless communication technologies. The experimental analysis demonstrated that background traffic negatively impacts latency and reliability in WSNs. In our analyses, the behavior of a 6TiSCH network based on single- and multi-hop topologies was evaluated by varying the amount of interfering Wi-Fi traffic. A mathematical model was proposed for both topologies to estimate the performance indicators. The model behaviour was compared with experimental data gathered on the real 6TiSCH devices. The OpenMote B device with the OpenWSN OS was selected to run the investigation aimed at collecting the realistic power consumption measurement. The comparison of the model data with the actual trial behavior showed that the model could estimate the WSN performance indicators granting high similarity rates. However, the mathematical model could not be used if the packet was waiting in the queue. This issue is not a constraint in many applications, since in most cases, the transmission rate is not higher than the one granted by the queue size available in the hardware. However, to face this issue, a TSCH-predictor was proposed to simulate the 6TiSCH protocol with a higher frequency to get a reasonable estimation of the WSN performance indicators. A further analysis was performed to compare the TSCH-predictor simulation results with real devices' experimental data. The study demonstrated that the simulation results shown by the TSCH-predictor are still very similar to the actual data performed on real devices. We hope that the proposed mathematical model and TSCH-predictor will help engineers design TSCH matrix parameters for WSNs with higher reliability and accuracy.

Appendix A

Publication List

Publications, including journals and international conference papers during the Ph.D. degree at Politecnico di Torino, are reported here.

Journal Papers

- M. Collotta, R. Ferrero, E. Giusto, M. Ghazi Vakili, J. Grecuccio, X. Kong, I. You, "A fuzzy control system for energy-efficient wireless devices in the Internet of vehicles", International Journal of Intelligent Systems, Vol. 36, Issue 4, p. 1595-1618, April. 2021,

  doi: 10.1002/int.22353

- S. Scanzio, M. Ghazi Vakili, G. Cena, C. G. Demartini, B. Montrucchio, A. Valenzano, C. Zunino, "Wireless sensor networks and TSCH: A compromise between reliability, power consumption, and latency", IEEE Access, vol. 8, p. 167042-167058, Sep. 2020, doi: 10.1109/ACCESS.2020.3022434

- B. Montrucchio, E. Giusto, M. Ghazi Vakili, S. Quer, R. Ferrero, C. Fornaro, "A Densely-Deployed, High Sampling Rate, Open-Source Air Pollution Monitoring WSN", IEEE Transactions on Vehicular Technology, vol. 69, p. 15786-15799, Nov. 2020,

  doi: 10.1109/TVT.2020.3035554

- G. Cena, C. G Demartini, M. Ghazi Vakili, S. Scanzio, A. Valenzano, C. Zunino, "Evaluating and modeling IEEE 802.15.4 TSCH resilience against Wi-Fi interference in new-generation highly-dependable wireless sensor networks", Ad Hoc Networks, vol. 106, p. 102199, Sep. 2020,

  doi: 10.1016/j.adhoc.2020.102199

- E. Giusto, M. Ghazi Vakili, F. Gandino, C. Demartini, B. Montrucchio, "Quantum Pliers Cutting the Blockchain", IT Professional, vol. 22, p. 90-96, Nov. 2020,

  doi: 10.1109/MITP.2020.2974690

Conferences

- M. Ghazi Vakili, C. Demartini, M. Guerrera, B. Montrucchio, "Open Source Fog Architecture for Industrial IoT Automation Based on Industrial Protocols", 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), vol. 1, p. 570-578,

  doi: 10.1109/COMPSAC.2019.00088.

- M. Ghazivakili, C. Demartini, C. Zunino, "Industrial data-collector by enabling OPC-UA standard for Industry 4.0", 2018 14th IEEE International Workshop on Factory Communication Systems (WFCS), p. 1-8,

  doi: 10.1109/WFCS.2018.8402364

- M. Hemmatpour, M. Ghazivakili, B. Montrucchio, M. Rebaudengo, "DIIG: a distributed industrial IoT gateway", 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), p. 755-759,

  doi: 10.1109/COMPSAC.2017.110

- R. Ferrero, M. Ghazi Vakili, E. Giusto, M. Guerrera, V. Randazzo, "Ubiquitous fridge with natural language interaction", 2019 IEEE International Conference on RFID Technology and Applications (RFID-TA), p. 404-409,

  doi: 10.1109/RFID-TA.2019.8892025

Datasets

- B. Montrucchio, E. Giusto, M. Ghazi Vakili, S. Quer, R. Ferrero, C. Fornaro, "A Densely-Deployed, High Sampling Rate, Open-Source Air Pollution Monitoring WSN", IEEEDataPort, August. 2020,

  doi: https://dx.doi.org/10.21227/m4pb-g538

- S. Scanzio, M. Ghazi Vakili, G. Cena, C. G. Demartini, B. Montrucchio, A. Valenzano, C. Zunino, "Wireless Sensor Networks Dataset (TSCH a Compromise Between Reliability, Power Consumption, and Latency)", IEEEDataPort, Jan. 2021,

  doi: https://dx.doi.org/10.21227/fg62-bp39

Appendix B

Kaa and DIIG algorithm

B.1 Kaa IoT Platform

There are several IoT platforms which enable programmers to connect different devices to the Internet [154]–[156]. Kaa is an open source middleware platform for the IoT nodes in order to manage data in back-end infrastructure through a server and endpoint SDK components. Since the Kaa server provides all the back-end functionality needed to operate on large-scale data, the Kaa IoT solution has been chosen for the purpose of this work [157].

Kaa server requires NoSQL and SQL database instances to store endpoint data and metadata, respectively. It manages each node in a combination of Control, Operations, and Bootstrap services, as Fig. B.1 shows. NoSQL database can be co-located with Kaa nodes on the same machine.

Kaa Control service manages overall system data and sends notifications to Operations services. It also maintains an up-to-date list of available Operation services.

The primary role of the Operation service is to communicate with multiple endpoints concurrently. Operation services process the endpoint requests and send data to them.

The Kaa Bootstrap service sends the information about Operation service connection parameters to the endpoints. Kaa SDKs contain a pre-generated list of Bootstrap services available in the Kaa server so that endpoints can query Bootstrap services from this list to retrieve connection parameters for the currently available Operation services.

As Fig. B.1 shows, the IoT gateway proposed by this paper is integrated with the Kaa SDK in the Kaa endpoint to support Modbus and S7 communication protocol. In order to do so, generated Kaa SDK are modified properly to load the bootstrap from the IoT server, and handle the requests coming from the IoT nodes, so that data can then be sent to the IoT server.

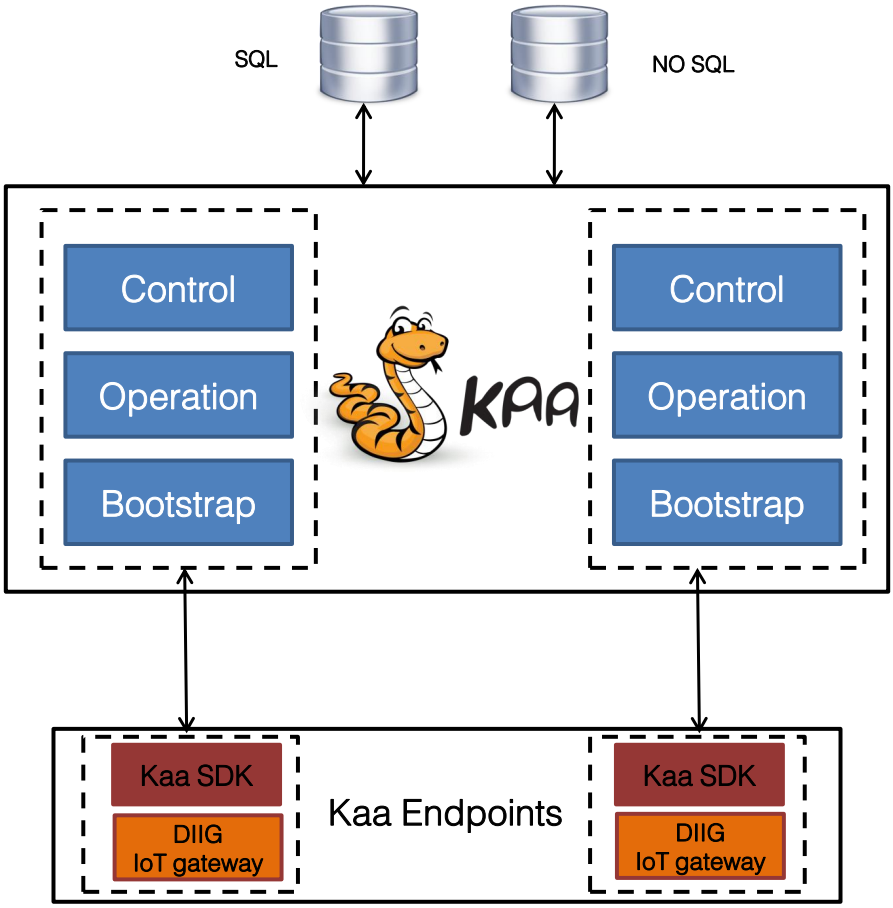


Figure B.1: Kaa IoT platform architecture integrated with DIIG gateway.

B.2 DIIG Protocol

The architecture proposed by this work is presented in Fig. B.2. It is based on the Kaa platform and consists of two main parts: the local protocol management and the Kaa SDK component. The local protocol management acquires data from a local network and sends them to the Kaa server. It can acquire data from the S7 or Modbus TCP networks which are protocols commonly used in the industrial environments. The SDK component is responsible for the communication with the Kaa server through the Kaa APIs.

There are two possible solutions to manage data transmission from the IoT nodes to the IoT gateway: solicited and unsolicited. In the solicited technique, the IoT gateway must visit each IoT node and if data are available, transmits it. This technique is not convenient since the IoT gateway should waste time sending pull data requests to the IoT nodes. Furthermore, this method is also rigid and unfair because an IoT node can saturate the traffic being thus responsible a nonuniform

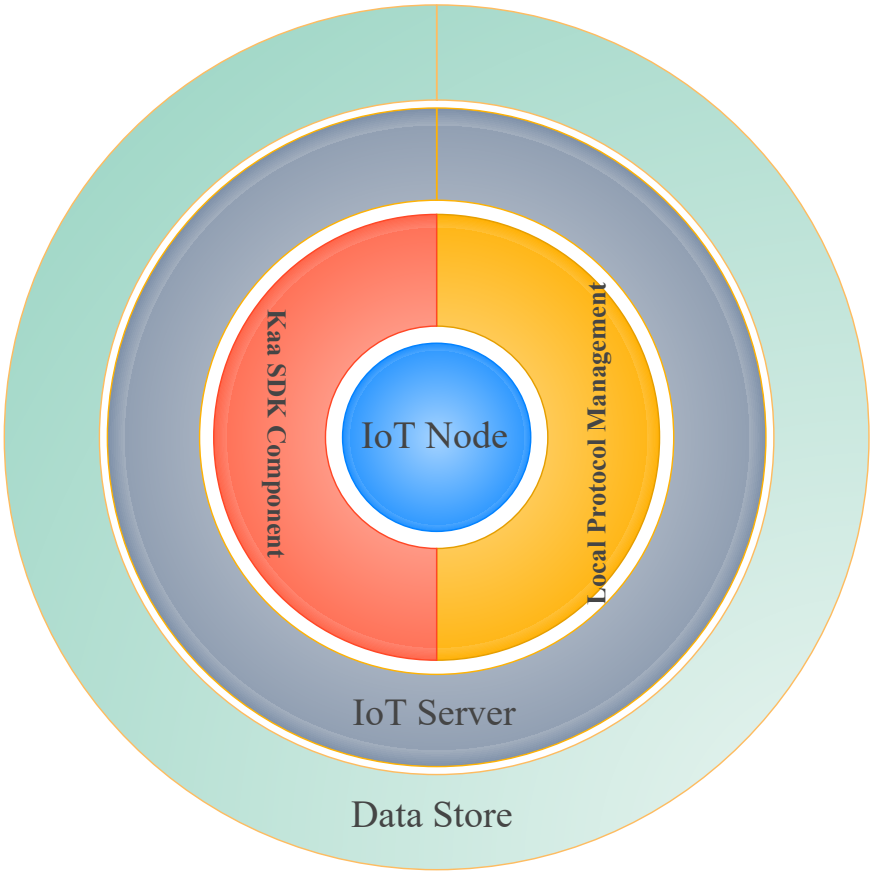


Figure B.2: DIIG architecture components.

traffic distribution.

However, unsolicited technique is more sophisticated in the sense that the IoT nodes send their data directly without any request coming from the IoT gateway. So, the overhead of sending pull data request from the IoT gateway is eliminated using this technique. The IoT gateway proposed in the following exploits this mechanism.

An important issue of this architecture is the communication between the local protocol management and the Kaa SDK component. As Fig. B.3 shows, the proposed gateway provides a parallel mechanism in order to achieve the real-time capability. For each connected IoT node of S7 or Modbus TCP networks a dedicated memory pool, a memory state table, two memory index containers, and two threads are created in the IoT gateway. The memory pool retains data transmitted from the IoT node. The size of the memory pool can be adjusted with the data transmission rate from the IoT node. The memory state table indicates the state of each memory address in the memory pool by signaling. Each memory address can adopt *busy* or *free* state. If the memory address is already occupied by a transmitted data from the IoT node its state is *busy*, otherwise it takes *free* state. The two memory index containers specify a free and a full index in the memory pool.

In the proposed communication protocol, while the IoT node generates its data, it reads *Free* index memory of the IoT gateway. Then, it lays its data in the read index in the memory pool. Afterwards, it toggles the corresponding memory state table. As can be seen in Fig. B.3, *Tr* thread always traverses the memory state table to find a free and a full memory addresses then fillings the *Free* and *Full* memory indices. The *Tr* traverses the memory state table in a round robin algorithm without any priority. The *Full* memory index is exploited by *Ts* thread to send the appropriate memory content to the Kaa server through Kaa APIs. Then, it toggles the corresponding memory state table. In order to avoid operating on the same data, if *Ts* reads the same *Full* memory index consecutively, it ignores the index till an update.

The dashed and red color boxes indicate the shared memories in the architecture,while the blue and green lines specify the read and write operations on the same memories. In order to orchestrate the local IoT node read and write operations on the shared memories, a spinlock is adopted. To handle remote read and write operations on the shared memories, Modbus TCP library has been modified properly according to the specific requirements related to the access to a shared resource.

In order to avoid data loss, each IoT node contains a local buffer. If the IoT node reads the same *Free* index buffer consecutively, it assumes that the IoT gateway is busy and stores its data in the local buffer. Then, it reads the *Free* index repeatedly till any modification occurs, at that point it sends its data immediately in the new index address. Each IoT node has 8 different channel blocks/registers to send its data. In order to avoid overwriting the data blocks, a synchronous function is

executed in the IoT node. Synchronous functions are often called blocking functions because they block the execution of the caller program and continues only when its job is completed.

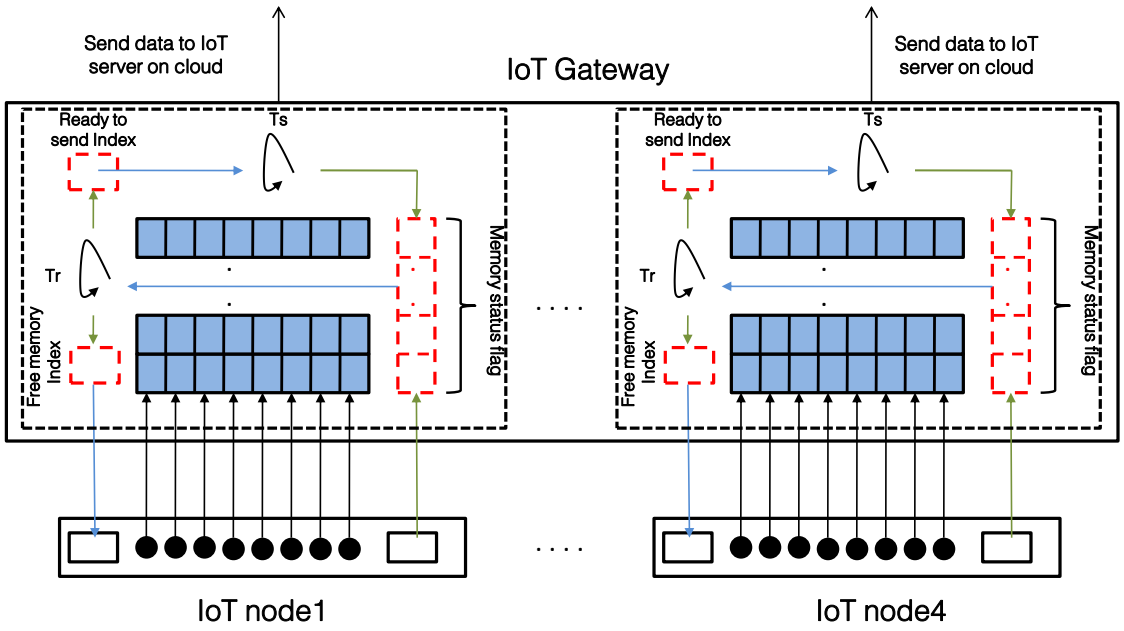


Figure B.3: DIIG architecture elements.

Bibliography

[1] T. Kramp, R. van Kranenburg, and S. Lange, "Introduction to the Internet of Things", in *Enabling Things to Talk*, A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. van Kranenburg, S. Lange, and S. Meissner, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–10, ISBN: 9783642404030. DOI: 10.1007/978-3-642-40403-0{_}1.

[2] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0", *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, Mar. 2017, ISSN: 1932-4529. DOI: 10.1109/MIE.2017.2649104.

[3] F. Shrouf, J. Ordieres, and G. Miragliotta, "Smart factories in Industry 4.0: A review of the concept and of energy management approached in production based on the Internet of Things paradigm", in *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, Dec. 2014, pp. 697–701. DOI: 10.1109/IEEM.2014.7058728.

[4] L. D. Xu, E. L. Xu, and L. Li, "Industry 4.0: state of the art and future trends", *International Journal of Production Research*, vol. 56, no. 8, pp. 2941–2962, 2018. DOI: 10.1080/00207543.2018.1444806. [Online]. Available: https://doi.org/10.1080/00207543.2018.1444806.

[5] Q. Qi and F. Tao, "Digital Twin and Big Data Towards Smart Manufacturing and Industry 4.0: 360 Degree Comparison", *IEEE Access*, vol. 6, pp. 3585–3593, 2018, ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2793265.

[6] A. B. de Sousa Jabbour, C. J. C. Jabbour, M. Godinho Filho, and D. Roubaud, "Industry 4.0 and the circular economy: a proposed research agenda and original roadmap for sustainable operations", *Annals of Operations Research*, vol. 270, no. 1-2, pp. 273–286, 2018, ISSN: 15729338. DOI: 10.1007/s10479-018-2772-8. [Online]. Available: https://doi.org/10.1007/s10479-018-2772-8.

[7] J. Lee, B. Bagheri, and H.-A. Kao, "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems", *Manufacturing Letters*, vol. 3, pp. 18–23, 2015, ISSN: 2213-8463. DOI: https://doi.org/10.1016/

j.mfglet.2014.12.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S221384631400025X.

[8] J. Rifkin, *The third industrial revolution: how lateral power is transforming energy, the economy, and the world*. Macmillan, 2011.

[9] S. Muntone, "Second Industrial Revolution", 2012.

[10] J. Robert E. Lucas, "Cambridge: Harvard University Press", ISBN 978-0-674-01601-9, 2002, pp. 109–110.

[11] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey", *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002, ISSN: 1389-1286. DOI: https://doi.org/10.1016/S1389-1286(01)00302-4.

[12] M. Erdelj, M. Król, and E. Natalizio, "Wireless Sensor Networks and Multi-UAV systems for natural disaster management", *Computer Networks*, vol. 124, pp. 72–86, 2017, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2017.05.021.

[13] T. Miyazaki, K. Anazawa, Y. Maruyama, S. Kobayashi, T. Segawa, and P. Li, "Resilient Information Management for Information Sharing in Disaster-Affected Areas Lacking Internet Access", in *Ad-Hoc, Mobile, and Wireless Networks*, M. R. Palattella, S. Scanzio, and S. Coleri Ergen, Eds., Cham: Springer International Publishing, 2019, pp. 3–17, ISBN: 978-3-030-31831-4.

[14] Y. Liao, M. Mollineaux, R. Hsu, R. Bartlett, A. Singla, A. Raja, R. Bajwa, and R. Rajagopal, "SnowFort: An Open Source Wireless Sensor Network for Data Analytics in Infrastructure and Environmental Monitoring", *IEEE Sensors Journal*, vol. 14, no. 12, pp. 4253–4263, Dec. 2014. DOI: 10.1109/JSEN.2014.2358253.

[15] B. Rashid and M. H. Rehmani, "Applications of wireless sensor networks for urban areas: A survey", *Journal of Network and Computer Applications*, vol. 60, pp. 192–219, 2016, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2015.09.008.

[16] X. Yu, P. Wu, W. Han, and Z. Zhang, "A survey on wireless sensor network infrastructure for agriculture", *Computer Standards & Interfaces*, vol. 35, no. 1, pp. 59–64, 2013, ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2012.05.001.

[17] M. Srbinovska, C. Gavrovski, V. Dimcev, A. Krkoleva, and V. Borozan, "Environmental parameters monitoring in precision agriculture using wireless sensor networks", *Journal of Cleaner Production*, vol. 88, pp. 297–307, 2015, ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2014.04.036.

[18] I. Chukwuemeka Chimsom and M. K. Habib, "Design of a Two-Tier WSN-based IoT Surveillance System with Cloud Integration", in *International Conference on Research and Education in Mechatronics (REM)*, May 2019, pp. 1–7. DOI: 10.1109/REM.2019.8744133.

[19] F. Civerchia, S. Bocchino, C. Salvadori, E. Rossi, L. Maggiani, and M. Petracca, "Industrial Internet of Things monitoring solution for advanced predictive maintenance applications", *Journal of Industrial Information Integration*, vol. 7, pp. 4–12, 2017, ISSN: 2452-414X. DOI: https://doi.org/10.1016/j.jii.2017.02.003.

[20] Z. Zhang, A. Mehmood, L. Shu, Z. Huo, Y. Zhang, and M. Mukherjee, "A Survey on Fault Diagnosis in Wireless Sensor Networks", *IEEE Access*, vol. 6, pp. 11 349–11 364, 2018. DOI: 10.1109/ACCESS.2018.2794519.

[21] B. Montrucchio, E. Giusto, M. Ghazi Vakili, S. Quer, R. Ferrero, and C. Fornaro, "A Densely-Deployed, High Sampling Rate, Open-Source Air Pollution Monitoring WSN", *IEEE Transactions on Vehicular Technology*, p. 1, Jan. 2020, ISSN: 1939-9359. DOI: 10.1109/TVT.2020.3035554.

[22] B. Velusamy and S. C. Pushpan, "An Enhanced Channel Access Method to Mitigate the Effect of Interference Among Body Sensor Networks for Smart Healthcare", *IEEE Sensors Journal*, vol. 19, no. 16, pp. 7082–7088, Aug. 2019, ISSN: 2379-9153. DOI: 10.1109/JSEN.2019.2913002.

[23] P. Zheng, H. wang, Z. Sang, R. Y. Zhong, Y. Liu, C. Liu, K. Mubarok, S. Yu, and X. Xu, "Smart manufacturing systems for Industry 4.0: Conceptual framework, scenarios, and future perspectives", *Frontiers of Mechanical Engineering*, vol. 13, no. 2, pp. 137–150, Jun. 2018, ISSN: 2095-0241. DOI: 10.1007/s11465-018-0499-5. [Online]. Available: https://doi.org/10.1007/s11465-018-0499-5.

[24] P. Adolphs, "Reference Architecture Model Industrie 4.0 (RAMI4.0)", Tech. Rep. July, 2015.

[25] C. Kaar, J. Frysak, and C. Stary, "Scaffolding RAMI4.0-Exploration as Design Support", pp. 1–8, 2018. DOI: 10.1145/3232078.3232098.

[26] V. Alcácer and V. Cruz-Machado, "Scanning the Industry 4.0: A Literature Review on Technologies for Manufacturing Systems", *Engineering Science and Technology, an International Journal*, vol. 22, no. 3, pp. 899–919, 2019, ISSN: 2215-0986. DOI: https://doi.org/10.1016/j.jestch.2019.01.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2215098618317750.

[27] J. Frysak, C. Kaar, and C. Stary, "Benefits and pitfalls applying RAMI4.0", *Proceedings - 2018 IEEE Industrial Cyber-Physical Systems, ICPS 2018*, pp. 32–37, 2018. DOI: 10.1109/ICPHYS.2018.8387633.

[28] J. Davis, "Smart Manufacturing", *Encyclopedia of Sustainable Technologies*, vol. 56, no. 1-2, pp. 417–427, 2017. DOI: 10.1016/B978-0-12-409548-9.10212-X. [Online]. Available: https://doi.org/10.1080/00207543.2017.1351644.

[29] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015, ISSN: 1553877X. DOI: 10.1109/COMST.2015.2444095.

[30] F. Banaie, J. Misic, V. B. Misic, M. H. Yaghmaee Moghaddam, and S. A. Hosseini Seno, "Performance analysis of multithreaded IoT gateway", *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3143–3155, 2019, ISSN: 23274662. DOI: 10.1109/JIOT.2018.2879467.

[31] C. C. Lin and J. W. Yang, "Cost-Efficient Deployment of Fog Computing Systems at Logistics Centers in Industry 4.0", *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4603–4611, 2018, ISSN: 15513203. DOI: 10.1109/TII.2018.2827920.

[32] M. Hemmatpour, M. Ghazivakili, B. Montrucchio, and M. Rebaudengo, "DIIG: A Distributed Industrial IoT Gateway", in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, Jul. 2017, pp. 755–759. DOI: 10.1109/COMPSAC.2017.110.

[33] International Standard, "IEC 62541-100:2015", Tech. Rep., 2015, p. 121. [Online]. Available: https://webstore.iec.ch/publication/21987.

[34] X. Ye and S. H. Hong, "Toward Industry 4.0 Components: Insights Into and Implementation of Asset Administration Shells", *IEEE Industrial Electronics Magazine*, vol. 13, no. 1, pp. 13–25, Mar. 2019, ISSN: 1932-4529. DOI: 10.1109/MIE.2019.2893397. [Online]. Available: https://ieeexplore.ieee.org/document/8673850/.

[35] A. ML, *{AutomationML} official website*, 2019. [Online]. Available: https://www.automationml.org/o.red.c/home.html.

[36] M. Mukherjee, L. Shu, and D. Wang, "Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges", *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 1826–1857, 2018, ISSN: 1553-877X. DOI: 10.1109/COMST.2018.2814571.

[37] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things", in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, New York, NY, USA: ACM, 2012, pp. 13–16, ISBN: 978-1-4503-1519-7. DOI: 10.1145/2342509.2342513. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513.

[38] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin, "Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges", *IEEE Access*, vol. 6, pp. 6505–6519, 2018, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2783682.

[39] I. Stojmenovic and S. Wen, "The Fog computing paradigm: Scenarios and security issues", in *2014 Federated Conference on Computer Science and Information Systems*, 2014, pp. 1–8. DOI: 10.15439/2014F503.

[40] S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues", in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata '15, New York, NY, USA: ACM, 2015, pp. 37–42, ISBN: 978-1-4503-3524-9. DOI: 10.1145/2757384.2757397. [Online]. Available: http://doi.acm.org/10.1145/2757384.2757397.

[41] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues", *Journal of Network and Computer Applications*, vol. 98, pp. 27–42, 2017, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.09.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517302953.

[42] Y. Lu and X. Xu, "Cloud-based manufacturing equipment and big data analytics to enable on-demand manufacturing services", *Robotics and Computer-Integrated Manufacturing*, vol. 57, pp. 92–102, 2019, ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2018.11.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0736584518302801.

[43] A. Y. Alqahtani, S. M. Gupta, and K. Nakashima, "Warranty and maintenance analysis of sensor embedded products using internet of things in industry 4.0", *International Journal of Production Economics*, vol. 208, pp. 483–499, 2019, ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2018.12.022. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925527318305000.

[44] N. Mohamed, J. Al-Jaroodi, and S. Lazarova-Molnar, "Leveraging the Capabilities of Industry 4.0 for Improving Energy Efficiency in Smart Factories", *IEEE Access*, vol. 7, pp. 18 008–18 020, 2019. DOI: 10.1109/ACCESS.2019.2897045.

[45] J. A. Saucedo-Martínez, M. Pérez-Lara, J. A. Marmolejo-Saucedo, T. E. Salais-Fierro, and P. Vasant, "Industry 4.0 framework for management and operations: a review", *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 3, pp. 789–801, 2018, ISSN: 18685145. DOI: 10.1007/s12652-017-0533-1.

[46] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues", *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, 2017, ISSN: 2452414X. DOI: 10.1016/j.jii.2017.04.005. [Online]. Available: http://dx.doi.org/10.1016/j.jii.2017.04.005.

[47] L. Wang, M. Törngren, and M. Onori, "Current status and advancement of cyber-physical systems in manufacturing", *Journal of Manufacturing Systems*, vol. 37, pp. 517–527, 2015, ISSN: 02786125. DOI: 10.1016/j.jmsy.2015.04.008. [Online]. Available: http://dx.doi.org/10.1016/j.jmsy.2015.04.008.

[48] *RAMI4.0 Standard*. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/DE/Anwendungsbeispiele/265-agentenbasierte-vernetzung-von-cyber-physischen-produktionssystemen-tu-muenchen/agentenbasierte-vernetzung-von-cyber-physischen-produktionssystemen.html.

[49] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Cla, "Middleware for internet of things: A survey", *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016, ISSN: 23274662. DOI: 10.1109/JIOT.2015.2498900.

[50] R. Zgheib, E. Conchon, and R. Bastide, "Semantic Middleware Architectures for IoT Healthcare Applications", in *Enhanced Living Environments: Algorithms, Architectures, Platforms, and Systems*, I. Ganchev, N. M. Garcia, C. Dobre, C. X. Mavromoustakis, and R. Goleva, Eds., Cham: Springer International Publishing, 2019, pp. 263–294. DOI: 10.1007/978-3-030-10752-9{_}11.

[51] M. A. A. da Cruz, J. J. P. C. Rodrigues, A. K. Sangaiah, J. Al-Muhtadi, and V. Korotaev, "Performance evaluation of IoT middleware", *Journal of Network and Computer Applications*, vol. 109, no. February, pp. 53–65, 2018, ISSN: 10958592. DOI: 10.1016/j.jnca.2018.02.013. [Online]. Available: https://doi.org/10.1016/j.jnca.2018.02.013.

[52] Y. Liu and X. Xu, "Industry 4.0 and Cloud Manufacturing: A Comparative Analysis", *Journal of Manufacturing Science and Engineering*, vol. 139, no. 3, 2016, ISSN: 1087-1357. DOI: 10.1115/1.4034667. [Online]. Available: https://doi.org/10.1115/1.4034667.

[53] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Chapter 4 - Fog Computing: principles, architectures, and applications", in *Internet of Things*, R. Buyya and A. V. Dastjerdi, Eds., Morgan Kaufmann, 2016, pp. 61–75, ISBN: 978-0-12-805395-9. DOI: https://doi.org/10.1016/B978-0-12-805395-9.00004-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128053959000046.

[54] I. Stojmenovic, "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks", in *2014 Australasian Telecommunication Networks and Applications Conference (ATNAC)*, Nov. 2014, pp. 117–122. DOI: 10.1109/ATNAC.2014.7020884.

[55] A. Davoudian, L. Chen, and M. Liu, "A Survey on NoSQL Stores", *ACM Comput. Surv.*, vol. 51, no. 2, 40:1–40:43, Apr. 2018, ISSN: 0360-0300. DOI: 10.1145/3158661. [Online]. Available: http://doi.acm.org/10.1145/3158661.

[56] V. Reniers, D. V. Landuyt, A. Rafique, and W. Joosen, "Object to NoSQL Database Mappers (ONDM): A systematic survey and comparison of frameworks", *Information Systems*, vol. 85, pp. 1–20, 2019, ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2019.05.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437918304290.

[57] P. Pääkkönen and D. Pakkala, "Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems", *Big Data Research*, vol. 2, no. 4, pp. 166–186, 2015, ISSN: 2214-5796. DOI: https://doi.org/10.1016/j.bdr.2015.01.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214579615000027.

[58] S. Vitturi, C. Zunino, and T. Sauter, "Industrial Communication Systems and Their Future Challenges: Next-Generation Ethernet, IIoT, and 5G", *Proceedings of the IEEE*, vol. 107, no. 6, pp. 944–961, Jun. 2019, ISSN: 0018-9219. DOI: 10.1109/JPROC.2019.2913443.

[59] Profinet, *Profinet*, \url{https://www.profibus.com/technology/profinet/}.

[60] Modbus, *Modbus protocol*, \url{http://modbus.org/}.

[61] O. P. C. Foundation, *OPC UA system*, \url{https://opcfoundation.org/about/what-is-opc/}.

[62] MQTT.org, *MQTT protocol*, \url{http://mqtt.org/faq}.

[63] Y. Zhang and W. W. Li, "Energy Consumption Analysis of a Duty Cycle Wireless Sensor Network Model", *IEEE Access*, vol. 7, pp. 33 405–33 413, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2903303.

[64] H. Yetgin, K. T. K. Cheung, M. El-Hajjar, and L. H. Hanzo, "A Survey of Network Lifetime Maximization Techniques in Wireless Sensor Networks", *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 828–854, 2017.

[65] M. Huang, K. Zhang, Z. Zeng, T. Wang, and Y. Liu, "An AUV-assisted Data Gathering Scheme based on Clustering and Matrix Completion for Smart Ocean", *IEEE Internet of Things Journal*, p. 1, 2020, ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2988035.

[66] Z. Li, Y. Liu, A. Liu, S. Wang, and H. Liu, "Minimizing Convergecast Time and Energy Consumption in Green Internet of Things", *IEEE Transactions on Emerging Topics in Computing*, p. 1, 2018, ISSN: 2168-6750. DOI: 10.1109/TETC.2018.2844282.

[67] M. Peng, W. Liu, T. Wang, and Z. Zeng, "Relay selection joint consecutive packet routing scheme to improve performance for wake-up radio-enabled WSNs", *Wireless Communications and Mobile Computing*, vol. 2020, pp. 1–32, Jan. 2020, ISSN: 1530-8669. DOI: 10.1155/2020/7230565.

[68] G. Cena, S. Scanzio, and A. Valenzano, "Improving Effectiveness of Seamless Redundancy in Real Industrial Wi-Fi Networks", *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2095–2107, May 2018, ISSN: 1941-0050. DOI: 10.1109/TII.2017.2759788.

[69] ——, "Experimental Evaluation of Techniques to Lower Spectrum Consumption in Wi-Red", *IEEE Transactions on Wireless Communications*, vol. 18, no. 2, pp. 824–837, Feb. 2019, ISSN: 1558-2248. DOI: 10.1109/TWC.2018.2884914.

[70] V. N. Swamy, P. Rigge, G. Ranade, B. Nikolić, and A. Sahai, "Wireless Channel Dynamics and Robustness for Ultra-Reliable Low-Latency Communications", *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 4, pp. 705–720, 2019.

[71] G. Cena, S. Scanzio, L. Seno, and A. Valenzano, "Comparison of Mixed Diversity Schemes to Enhance Reliability of Wireless Networks", in *Ad-Hoc, Mobile, and Wireless Networks*, M. R. Palattella, S. Scanzio, and S. Coleri Ergen, Eds., Cham: Springer International Publishing, 2019, pp. 118–135, ISBN: 978-3-030-31831-4.

[72] IEEE, "IEEE Standard for Low-Rate Wireless Networks", *IEEE Std 802.15.4-2015 (Rev. of IEEE Std 802.15.4-2011)*, pp. 1–709, Apr. 2016. DOI: 10.1109/IEEESTD.2016.7460875.

[73] "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)*, pp. 1–3534, Dec. 2016. DOI: 10.1109/IEEESTD.2016.7786995.

[74] R. Koutsiamanis, G. Z. Papadopoulos, X. Fafoutis, J. M. D. Fiore, P. Thubert, and N. Montavont, "From Best Effort to Deterministic Packet Delivery for Wireless Industrial IoT Networks", *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4468–4480, Oct. 2018. DOI: 10.1109/TII.2018.2856884.

[75] G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Experimental Analysis and Comparison of Industrial IoT Devices based on TSCH", in *24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2019)*, 2019, pp. 184–191. DOI: 10.1109/ETFA.2019.8869410.

[76] ISO/IEC, "Information technology - Home electronic system (HES) architecture - Part 3-11: Frequency modulated wireless short-packet (FMWSP) protocol optimised for energy harvesting - Architecture and lower layer protocols", International Organization for Standardization/International Electrotechnical Commission, Tech. Rep. ISO/IEC 14543-3-11:2016, pp. 1–25.

[77] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks", in *International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Apr. 2004, pp. 224–. DOI: 10.1109/IPDPS.2004.1303264.

[78] R. Piyare, A. L. Murphy, C. Kiraly, P. Tosato, and D. Brunelli, "Ultra Low Power Wake-Up Radios: A Hardware and Networking Survey", *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2117–2157, 2017, ISSN: 2373-745X. DOI: 10.1109/COMST.2017.2728092.

[79] M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel, "On Optimal Scheduling in Duty-Cycled Industrial IoT Applications Using IEEE802.15.4e TSCH", *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3655–3666, 2013.

[80] M. R. Palattella, T. Watteyne, Q. Wang, K. Muraoka, N. Accettura, D. Dujovne, L. A. Grieco, and T. Engel, "On-the-Fly Bandwidth Reservation for 6TiSCH Wireless Industrial Networks", *IEEE Sensors Journal*, vol. 16, no. 2, pp. 550–560, 2016.

[81] L. L. Bello, A. Lombardo, S. Milardo, G. Patti, and M. Reno, "Software-Defined Networking for Dynamic Control of Mobile Industrial Wireless Sensor Networks", in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 290–296. DOI: 10.1109/ETFA.2018.8502457.

[82] H. Nishimoto, Y. Kawahara, and T. Asami, "Prototype implementation of ambient RF energy harvesting wireless sensor networks", in *IEEE SENSORS*, Nov. 2010, pp. 1282–1287. DOI: 10.1109/ICSENS.2010.5690588.

[83] F. K. Shaikh and S. Zeadally, "Energy harvesting in wireless sensor networks: A comprehensive review", *Renewable and Sustainable Energy Reviews*, vol. 55, pp. 1041–1054, 2016, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2015.11.010.

[84] S. Y. Shin, H. S. Park, and W. H. Kwon, "Mutual interference analysis of IEEE 802.15.4 and IEEE 802.11b", *Computer Networks*, vol. 51, no. 12, pp. 3338–3353, 2007, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2007.01.034.

[85] J. Kim, W. Jeon, K. Park, and J. P. Choi, "Coexistence of Full-Duplex-Based IEEE 802.15.4 and IEEE 802.11", *IEEE Trans. Ind. Informat.*, vol. 14, no. 12, pp. 5389–5399, Dec. 2018, ISSN: 1551-3203. DOI: 10.1109/TII.2018.2866307.

[86] S. Pollin, I. Tan, B. Hodge, C. Chun, and A. Bahai, "Harmful Coexistence Between 802.15.4 and 802.11: A Measurement-based Study", in *3rd International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CrownCom)*, May 2008, pp. 1–6. DOI: 10.1109/CROWNCOM.2008.4562460.

[87] L. Angrisani, M. Bertocco, D. Fortin, and A. Sona, "Assessing coexistence problems of IEEE 802.11b and IEEE 802.15.4 wireless networks through cross-layer measurements", in *IEEE Instrumentation Measurement Technology Conference (IMTC)*, May 2007, pp. 1–6. DOI: 10.1109/IMTC.2007.379454.

[88] M. Petrova, L. Wu, P. Mahonen, and J. Riihijarvi, "Interference Measurements on Performance Degradation between Colocated IEEE 802.11g/n and IEEE 802.15.4 Networks", in *Sixth International Conference on Networking (ICN 2007)*, Apr. 2007, p. 93. DOI: 10.1109/ICN.2007.53.

[89] B. Polepalli, W. Xie, D. Thangaraja, M. Goyal, H. Hosseini, and Y. Bashir, "Impact of IEEE 802.11n Operation on IEEE 802.15.4 Operation", in *2009 Int. Conference on Advanced Information Networking and Applications Workshops*, May 2009, pp. 328–333. DOI: 10.1109/WAINA.2009.102.

[90] F. Yao, S. Yang, and W. Zheng, "Mitigating interference caused by IEEE 802.11b in the IEEE 802.15.4 WSN within the environment of smart house", in *IEEE International Conference on Systems, Man and Cybernetics*, Oct. 2010, pp. 2800–2807. DOI: 10.1109/ICSMC.2010.5641899.

[91] S. Ben Yaala, F. Théoleyre, and R. Bouallegue, "Performance study of co-located IEEE 802.15.4-TSCH networks: Interference and coexistence", in *IEEE Symposium on Computers and Communication (ISCC)*, Jun. 2016, pp. 513–518. DOI: 10.1109/ISCC.2016.7543790.

[92] J. Umer, H. Di, L. Peilin, and Y. Yueming, "Frequency hopping in IEEE 802.15.4 to mitigate IEEE 802.11 interference and fading", *Journal of Systems Engineering and Electronics*, vol. 29, no. 3, pp. 445–455, Jun. 2018. DOI: 10.21629/JSEE.2018.03.01.

[93] S. Zoppi, H. M. Gürsu, M. Vilgelm, and W. Kellerer, "Reliable hopping sequence design for highly interfered wireless sensor networks", in *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN 2017)*, Jun. 2017, pp. 1–7. DOI: 10.1109/LANMAN.2017.7972164.

[94] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function", *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, pp. 535–547, Mar. 2000, ISSN: 1558-0008. DOI: 10.1109/49.840210.

[95] F. Babich and M. Comisso, "Theoretical analysis of asynchronous multipacket reception in 802.11 networks", *IEEE Transactions on Communications*, vol. 58, no. 6, pp. 1782–1794, 2010. DOI: 10.1109/TCOMM.2010.06.09,ISSN={\{}1558-0857{\}},month={\{}June{\}},.

[96] P. P. Pham, "Comprehensive Analysis of the IEEE 802.11", *Mobile Networks and Applications*, vol. 10, no. 5, pp. 691–703, Oct. 2005, ISSN: 1572-8153. DOI: 10.1007/s11036-005-3363-x.

[97] D. De Guglielmo, B. Al Nahas, S. Duquennoy, T. Voigt, and G. Anastasi, "Analysis and Experimental Evaluation of IEEE 802.15.4e TSCH CSMA-CA Algorithm", *IEEE Transactions on Vehicular Technology*, vol. 66, no. 2, pp. 1573–1588, Feb. 2017, ISSN: 1939-9359. DOI: 10.1109/TVT.2016.2553176.

[98] C. Ouanteur, L. Bouallouche-Medjkoune, and D. Aïssani, "An Enhanced Analytical Model and Performance Evaluation of the IEEE 802.15.4e TSCH CA", *Wireless Personal Communications*, vol. 96, no. 1, pp. 1355–1376, Sep. 2017, ISSN: 1572-834X. DOI: 10.1007/s11277-017-4241-0.

[99] P. Luong, T. M. Nguyen, and L. B. Le, "Throughput analysis for coexisting IEEE 802.15.4 and 802.11 networks under unsaturated traffic", *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, p. 127, May 2016, ISSN: 1687-1499. DOI: 10.1186/s13638-016-0586-4.

[100] S. Y. Shin, "Throughput analysis of IEEE 802.15.4 network under IEEE 802.11 network interference", *AEU - International Journal of Electronics and Communications*, vol. 67, no. 8, pp. 686–689, 2013, ISSN: 1434-8411. DOI: https://doi.org/10.1016/j.aeue.2013.02.007.

[101] A. Nikoukar, S. Raza, A. Poole, M. Güneş, and B. Dezfouli, "Low-Power Wireless for the Internet of Things: Standards and Applications", *IEEE Access*, vol. 6, pp. 67 893–67 926, 2018, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2879189.

[102] P. Gaj, S. Scanzio, and L. Wisniewski, "Guest Editorial: Heterogeneous Industrial Networks of the Current and Next-Generation Factories", *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5539–5542, 2020.

[103] P. Thubert, "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4", *IETF Std draft-ietf-6tisch-architecture-19*, pp. 1–60, Dec. 2018.

[104] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: A Tutorial", *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 595–615, 2020.

[105] X. Vilajosana, T. Watteyne, M. Vučinić, T. Chang, and K. S. J. Pister, "6TiSCH: Industrial Performance for IPv6 Internet-of-Things Networks", *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1153–1165, 2019.

[106] M. Vučinić, T. Chang, B. Škrbić, E. Kočan, M. Pejanović-Djurišić, and T. Watteyne, "Key Performance Indicators of the Reference 6TiSCH Implementation in Internet-of-Things Scenarios", *IEEE Access*, vol. 8, pp. 79 147–79 157, 2020.

[107] "IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer", *IEEE Std 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011)*, pp. 1–225, 2012.

[108] D. Stanislowski, X. Vilajosana, Q. Wang, T. Watteyne, and K. S. J. Pister, "Adaptive Synchronization in IEEE802.15.4e Networks", *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 795–802, 2014.

[109] M. Mongelli and S. Scanzio, "A neural approach to synchronization in wireless networks with heterogeneous sources of noise", *Ad Hoc Networks*, vol. 49, no. 1, S. C. Mukhopadhyay, Ed., pp. 1–16, Oct. 2016, ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2016.06.002. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1570870516301366.

[110] Q. Wang, K. Jaffrès-Runser, Y. Xu, J. Scharbarg, Z. An, and C. Fraboul, "TDMA Versus CSMA/CA for Wireless Multihop Communications: A Stochastic Worst-Case Delay Analysis", *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 877–887, Apr. 2017, ISSN: 1551-3203. DOI: 10.1109/TII.2016.2620121.

[111] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey", *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, 2009, ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2008.06.003.

[112] M. R. Palattella, P. Thubert, X. Vilajosana, T. Watteyne, Q. Wang, and T. Engel, "6TiSCH Wireless Industrial Networks: Determinism Meets IPv6", in *Internet of Things: Challenges and Opportunities*, S. C. Mukhopadhyay, Ed., Cham: Springer International Publishing, 2014, pp. 111–141, ISBN: 978-3-319-04223-7. DOI: 10.1007/978-3-319-04223-7{_}5.

[113] N. Taheri Javan, M. Sabaei, and V. Hakami, "IEEE 802.15.4.e TSCH-Based Scheduling for Throughput Optimization: A Combinatorial Multi-Armed Bandit Approach", *IEEE Sensors Journal*, vol. 20, no. 1, pp. 525–537, 2020.

[114] A. Elsts, S. Kim, H. Kim, and C. Kim, "An Empirical Survey of Autonomous Scheduling Methods for TSCH", *IEEE Access*, vol. 8, pp. 67 147–67 165, 2020.

[115] A. Karaagac, I. Moerman, and J. Hoebeke, "Hybrid Schedule Management in 6TiSCH Networks: The Coexistence of Determinism and Flexibility", *IEEE Access*, vol. 6, pp. 33 941–33 952, 2018.

[116] A. Elsts, X. Fafoutis, G. Oikonomou, R. Piechocki, and I. Craddock, "TSCH Networks for Health IoT: Design, Evaluation, and Trials in the Wild", *ACM Trans. Internet Things*, vol. 1, no. 2, Apr. 2020, ISSN: 2691-1914. DOI: 10.1145/3366617. [Online]. Available: https://doi.org/10.1145/3366617.

[117] Q. Wang, X. Vilajosana, and T. Watteyne, "6TiSCH Operation Sublayer (6top) Protocol (6P)", *IETF RFC 8480*, pp. 1–50, Nov. 2018.

[118] P. Thubert, T. Winter, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks", *IETF RFC 6550*, pp. 1–157, Mar. 2012.

[119] S. Yoo, P. K. Chong, D. Kim, Y. Doh, M. Pham, E. Choi, and J. Huh, "Guaranteeing Real-Time Services for Industrial Wireless Sensor Networks With IEEE 802.15.4", *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3868–3876, Nov. 2010, ISSN: 1557-9948. DOI: 10.1109/TIE.2010.2040630.

[120] C.-S. Chen and D.-S. Lee, "Energy Saving Effects of Wireless Sensor Networks: A Case Study of Convenience Stores in Taiwan", *Sensors*, vol. 11, no. 2, pp. 2013–2034, Feb. 2011, ISSN: 1424-8220. DOI: 10.3390/s110202013.

[121] G. Acar and A. E. Adams, "ACMENet: an underwater acoustic sensor network protocol for real-time environmental monitoring in coastal areas", *IEEE Proceedings - Radar, Sonar and Navigation*, vol. 153, no. 4, pp. 365–380, Aug. 2006, ISSN: 1350-2395. DOI: 10.1049/ip-rsn:20045060.

[122] G. Cena, C. G. Demartini, M. Ghazi Vakili, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluating and Modeling IEEE 802.15.4 TSCH Resilience against Wi-Fi Interference in New-Generation Highly-Dependable Wireless Sensor Networks", *Ad Hoc Networks*, vol. 106, p. 102 199, 2020, ISSN: 1570–8705. DOI: https://doi.org/10.1016/j.adhoc.2020.102199.

[123] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)", *IETF RFC 7252*, pp. 1–111, Jun. 2014.

[124] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks", in *International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04, New York, NY, USA: ACM, 2004, pp. 95–107, ISBN: 1-58113-879-2. DOI: 10.1145/1031495.1031508.

[125] X. Vilajosana, Q. Wang, F. Chraim, T. Watteyne, T. Chang, and K. S. J. Pister, "A Realistic Energy Consumption Model for TSCH Networks", *IEEE Sensors Journal*, vol. 14, no. 2, pp. 482–489, Feb. 2014, ISSN: 2379-9153. DOI: 10.1109/JSEN.2013.2285411.

[126] M. Ghazivakili, C. Demartini, and C. Zunino, "Industrial data-collector by enabling OPC-UA standard for Industry 4.0", in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, Jun. 2018, pp. 1–8. DOI: 10.1109/WFCS.2018.8402364.

[127] VDI/VDE, "Status Report - The Reference Architectural Model Industrie 4.0 (RAMI4.0)", *Vdi/Vde,Zvei*, vol. 0, no. April, 2015.

[128] P. Ferrari, A. Flammini, S. Rinaldi, E. Sisinni, D. Maffei, and M. Malara, "Evaluation of Communication Delay in IOT Applications Based on OPC UA", *2018 Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2018 - Proceedings*, pp. 224–229, 2018. DOI: 10.1109/METROI4.2018.8428346.

[129] M. Ghazi Vakili, C. Demartini, M. Guerrera, and B. Montrucchio, "Open Source Fog Architecture for Industrial IoT Automation Based on Industrial Protocols", in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2019, pp. 570–578. DOI: 10.1109/COMPSAC.2019.00088.

[130] W. Steiner and S. Poledna, "Fog computing as enabler for the Industrial Internet of Things", *e & i Elektrotechnik und Informationstechnik*, vol. 133, no. 7, pp. 310–314, Nov. 2016, ISSN: 1613-7620. DOI: 10.1007/s00502-016-0438-2. [Online]. Available: https://doi.org/10.1007/s00502-016-0438-2.

[131] C. C. Byers, "Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks", *IEEE Communications Magazine*, vol. 55, no. 8, pp. 14–20, Aug. 2017, ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1600885.

[132] *Develop with Docker Engine SDKs and API*, Jan. 2019. [Online]. Available: https://docs.docker.com/develop/sdk/.

[133] *Node-RED*, Jan. 2019. [Online]. Available: https://nodered.org/.

[134] *IBM Emerging Technology*, Jan. 2019. [Online]. Available: https://emerging-technology.co.uk/.

[135] *JS Foundation*, Jan. 2019. [Online]. Available: https://js.foundation/.

[136] *Grafana - The open platform for analytics and monitoring*, Jan. 2019. [Online]. Available: https://grafana.com/.

[137] B. Z. Cadersaib, H. B. Sta, and B. A. G. Rahimbux, "Making an Interoperability Approach between ERP and Big Data Context", in *2018 Sixth International Conference on Enterprise Systems (ES)*, Oct. 2018, pp. 146–153. DOI: 10.1109/ES.2018.00030.

[138] J. R. Farr and M. H. Jawad, "Design of Heat Exchangers", in, New York, NY: ASME, 2006, ch. Design of, ISBN: 0791802396. [Online]. Available: http://dx.doi.org/10.1115/1.802396.ch7.

[139] S. Scanzio, M. Ghazi Vakili, G. Cena, C. G. Demartini, B. Montrucchio, A. Valenzano, and C. Zunino, *Wireless Sensor Networks Dataset (TSCH a Compromise Between Reliability, Power Consumption, and Latency)*, 2020. DOI: 10.21227/fg62-bp39. [Online]. Available: https://dx.doi.org/10.21227/fg62-bp39.

[140] A. Conta, S. Deering, and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", *IETF RFC 4443*, pp. 1–23, Mar. 2006.

[141] G. Cena, I. C. Bertolotti, A. Valenzano, and C. Zunino, "Evaluation of Response Times in Industrial WLANs", *IEEE Transactions on Industrial Informatics*, vol. 3, no. 3, pp. 191–201, Aug. 2007. DOI: 10.1109/TII.2007.903219.

[142] T. Watteyne, A. Mehta, and K. Pister, "Reliability through frequency diversity", in *Proceedings of the 6th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks - PE-WASUN '09*, ser. PE-WASUN '09, New York, New York, USA: ACM Press, 2009, p. 116, ISBN: 9781605586182. DOI: 10.1145/1641876.1641898. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1641876.1641898.

[143] P. Du and G. Roussos, "Spectrum-aware wireless sensor networks", in *IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC 2013)*, 2013, pp. 2321–2325. DOI: 10.1109/PIMRC.2013.6666532.

[144] *OpenWSN*, \url{https://openwsn.atlassian.net/wiki/}.

[145] *OpenMote*, \url{https://www.industrialshields.com/open-mote-b- industrialshields-open-source-device-ready-for-internet -of-things}.

[146] S. Scanzio, M. G. Vakili, G. Cena, C. G. Demartini, B. Montrucchio, A. Valenzano, and C. Zunino, "Wireless Sensor Networks and TSCH: A Compromise Between Reliability, Power Consumption, and Latency", *IEEE Access*, vol. 8, pp. 167 042–167 058, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3022434.

[147] R. Tavakoli, M. Nabi, T. Basten, and K. Goossens, "Topology Management and TSCH Scheduling for Low-Latency Convergecast in In-Vehicle WSNs", *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1082–1093, 2019.

[148] G. Cena, S. Scanzio, L. Seno, A. Valenzano, and C. Zunino, "Energy-Efficient Link Capacity Overprovisioning In Time Slotted Channel Hopping Networks", in *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, 2020, pp. 1–8.

[149] L. Seno, G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Enhancing Communication Determinism in Wi-Fi Networks for Soft Real-Time Industrial Applications", *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 866–876, 2017.

[150] G. Cena, S. Scanzio, and A. Valenzano, "SDMAC: A Software-Defined MAC for Wi-Fi to Ease Implementation of Soft Real-Time Applications", *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3143–3154, 2019.

[151] E. Municio, G. Daneels, M. Vučinić, S. Latré, J. Famaey, Y. Tanaka, K. Brun, K. Muraoka, X. Vilajosana, and T. Watteyne, "Simulating 6TiSCH networks", *Transactions on Emerging Telecommunications Technologies*, vol. 30, no. 3, e3494, 2019. DOI: https://doi.org/10.1002/ett.3494. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3494.

[152] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors", in *29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 455–462. DOI: 10.1109/LCN.2004.38.

[153] A. Elsts, "TSCH-Sim: Scaling Up Simulations of TSCH and 6TiSCH Networks", *Sensors*, vol. 20, no. 19, 2020, ISSN: 1424-8220. DOI: 10.3390/s20195663. [Online]. Available: https://www.mdpi.com/1424-8220/20/19/5663.

[154] *mbed IoT Platform*, https://www.mbed.com/en/, Accessed: 2020-01-15.

[155] *platformio IoT Platform*, http://platformio.org/, Accessed: 2020-01-15.

[156] *iotivity IoT Platform*, https://iotivity.org/, Accessed: 2020-01-15.

[157] *Kaa IoT Platform*, https://www.kaaproject.org, Accessed: 2020-01-15.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.