# Deterministic Networking for the Industrial IoT

Keoma Brun-Laguna

Thesis

of the

École Doctorale Informatique, Télécommunications
et Électronique (Paris)

# Deterministic Networking for the Industrial IoT

presented by

Keoma BRUN-LAGUNA

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

in

Computer Science

at the

Université Pierre et Marie Curie

Presented on the 18th of December 2018.

Jury:

| | | |
|---|---|---|
| Fabrice Théoleyre | CNRS (ICube), Strasbourg, France | Reviewer |
| Sofie Pollin | KU Leuven, Leuven, Belgium | Reviewer |
| Lance Doherty | Analog Device, Cork, Ireland | Examiner |
| Marcelo Dias de Amorim | LiP6, Paris, France | Examiner |
| Pascale Minet | Inria, Paris, France | PhD Adviser |
| Thomas Watteyne | Inria, Paris, France | PhD co-Adviser |

# Abstract

The Internet of Things (IoT) evolved from a connected toaster in 1990 to networks of hundreds of tiny devices used in industrial applications. Those "Things" usually are tiny electronic devices able to measure a physical value (temperature, humidity, etc.) and/or to actuate on the physical world (pump, valve, etc). Due to their cost and ease of deployment, battery-powered wireless IoT networks are rapidly being adopted.

The promise of wireless communication is to offer wire-like connectivity. Major improvements have been made in that sense, but many challenges remain as industrial application have strong operational requirements. This section of the IoT application is called Industrial IoT (IIoT).

The main IIoT requirement is reliability. Every bit of information that is transmitted in the network must not be lost. Current off-the-shelf solutions offer over 99.999% reliability. That is, for every 100k packets of information generated, less than one is lost.

Then come latency and energy-efficiency requirements. As devices are battery-powered, they need to consume as little as possible to be able to operate during years. The next step for the IoT is to target time-critical applications.

Industrial IoT technologies are now adopted by companies over the world, and are now a proven solution. Yet, challenges remain and some of the limits of the technologies are still not fully understood. In this work we address TSCH-based Wireless Sensor Networks and study their latency and lifetime limits under real-world conditions.

We gathered 3M network statistics 32M sensor measurements on 11 datasets with a total of 170,037 mote hours in real-world and testbeds deployments. We assembled what we believed to be the largest dataset available to the networking community.

Based on those datasets and on insights we learned from deploying networks in real-world conditions, we study the limits and trade-offs of TSCH-based Wireless Sensor Networks. We provide methods and tools to estimate the network performances of such networks in various scenarios. We believe we assembled the right tools for protocol designer to built deterministic networking to the Industrial IoT.

# Résumé

L'Internet des Objets (IoT) a évolué d'un toaster connecté en 1990 vers des réseaux de centaines de petit appareils utilisés dans des applications industrielle. Ces « Objects » sont abituellement de petit appareils électroniques capable de mesurer une valeur physique (temperature, humidité, etc.) et/ou d'agir sur le monde physique (pump, valve, etc.). De part leur faible coût et leur facilité de déploiement, ces réseaux sans fil alimentés par batteries ont étés rapidement adoptés.

La promesse des communications sans fil est d'offrir une connectivité similaire au réseau fillaires. De nombreuses amélioration ont étés fait dans ce sens, mais plein de défis restent à surmonter car les applications industrielles ont de fortes exigences opérationnelles. Cette section de l'IoT s'appelle l'Internet Industriel des Objets.

La principale exigence est la fiabilité. Chaque bout d'information transmit dans le réseau ne doit pas être perdu. Des solutions commerciales sont aujourd'hui accessibles et propose des fiabilités de l'ordre de 99.999 %. C'est à dire, pour chaque centaine de packet d'information généré, moins d'un est perdu.

Vient ensuite la latence et l'efficience énergétique. Comme ces appareils sont alimentés par des batteries, ils doivent consommer le moins possible et être capable d'opérer pendant des années. La prochaine étape pour l'IoT est d'être appliqué au applications nécessitant des garanties de latence.

Les technologies de l'IIoT sont maintenant adoptés par de nombreuses entreprises autour du monde et sont maintenant des technologies éprouvées. Néanmoins des défis restent à accomplir et certaines limites de ces technologies ne sont pas encore connues.

Dans ce travail, nous nous adressons au réseaux sans fils fondés sur TSCH dont nous testons les limites de latence et de durée de vie dans des conditions réelles. Nous avons collecté plus de 3M statistiques réseaux et 32M données de capteurs dans 11 déploiements sur un total de 170,037 heures machines dans des environnements réels ainsi que dans des bancs d'essais.

Nous avons réuni ce que nous pensons être le plus grand jeu de données de réseau TSCH disponible à la communauté réseau. En s'appuyant sur ces données et sur notre experience des réseaux sans fils en milieu réel, nous avons étudié les limites des réseaux TSCH et avons fourni des méthodes et outils

qui permettent d'estimer des performances de ces réseaux dans diverses conditions. Nous pensons avoir assemblé les bons outils pour que les architectes de protocols réseaux construise des réseaux déterministes pour l'IIoT.

**Keywords**

Wireless;Sensors;Network;IoT;WSN;Industrial;

**Mot-Clés**

Réseaux sans-fils;Capteurs;IoT;IIoT;

# Contents

# Acronyms

| | |
|---|---|
| ACK | Acknowledgement (frame) |
| ARHO | American River Hydrologic Observatory |
| ASN | Absolute Slot Number |
| COAP | Constrained Application Protocol |
| CSMA/CA | Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA) |
| CTS | Clear to Send |
| CZO | Critical Zone Observatory |
| DetNet | Deterministic Networking |
| DODAG | Destination-Oriented Directed Acyclic Graph |
| ETSI | European Telecommunications Standards Institute |
| FDMA | Frequency-Division Multiple Access |
| FSPL | Free Space Path Loss |
| HART | Highway Addressable Remote Transducer |
| HR | Health Reports |
| HTTP | HyperText Transfer Protocol |
| IEEE | Institute of Electrical and Electronics Engineers |
| IETF | Internet Engineering Task Force |
| IIOT | Industrial Internet of Things |
| IOT | Internet of Things |
| IPC | Industrial Process Control |
| IPv6 | Internet Protocol version 6 |
| JSON | JavaScript Object Notation |
| KPI | Key Performance Indicators |
| LLN | Low-Power and Lossy Network |
| LOS | Line Of Sight |

| | |
|---|---|
| MTU | Maximum Transmission Unit |
| M2M | Machine-to-Machine |
| OQPSK | Offset quadrature phase-shift keying |
| OSI | Open Systems Interconnection |
| PDR | Packet Delivery Ratio |
| PLC | Power Line Carrier |
| ppm | parts per million |
| PRR | Packet Reception Rate |
| QoS | Quality of Service |
| RFC | Request For Comments |
| RFID | Radio Frequency Identifiers |
| RPL | Routing Protocol for Low-Power and Lossy Networks |
| RTS | Request to Send |
| SDO | Standards Developing Organizations |
| SNR | Signal-to-Noise Ratio |
| SOL | Sensor Object Library |
| TASA | Traffic Aware Scheduling Algorithm |
| TDM | Time Division Multiplexing |
| TDMA | Time Division Multiple Access |
| TLS | Transport Layer Security |
| TSCH | Time Slotted Channel Hopping |
| TSMP | Time Synchronized Mesh Protocol |
| UDP | User Datagram Protocol |
| WSN | Wireless Sensor Networks |
| 6LoWPAN | IPv6 over Low-Power Wireless Personal Area Networks |
| 6P | 6top Protocol |
| 6TiSCH | IPv6 over TSCH |
| 6top | 6TiSCH Operation Sublayer |

# Glossary

**Mercator** is a system to collect large wireless connectivity datasets that are dense in time, space and frequency. This system allows us to understand the quality of wireless connectivity, and its variation over time and frequency.

**SolSystem** is a set of tools we developed and used in our real-world deployment. SolSystem includes *(i)* software that runs next to the deployment gateway. It backs up the data locally, parses it and forwards it to the API. *(ii)* software that runs in a datacenter. This software parses, stores, analyzes, and displays the data.

**TSCH**: Time Slotted Channel Hopping (TSCH) is a medium access control method for shared medium networks. TSCH is designed for applications that require reliability and ultra long battery life.

**6TiSCH**: 6TiSCH is a working group at the IETF which is standardizing how to combine IEEE802.15.4 Time Slotted Channel Hopping (TSCH) with IPv6. In this work, I also use the terms *6TiSCH stack* and *6TiSCH network* to refer to the stack of networking protocols defined by the working group, and a network using that 6TiSCH stack, respectively.

# Acknowledgements

The thesis work was made possible thanks to National Institute of Research in Computing and Automation (Inria), and Doctoral School of the University Pierre et Marie Curie (UPMC) in Paris. I would like to send my deepest appreciation to my advisor Thomas Watteyne, for the continuous support, patience, and encouragement he gave through my work. Without him I would not had the chance to work with such competent team across the world, and to experience the cutting edge of technology in such great conditions. He always gave me the motivation and advice I needed. I deeply thank my adviser Pascale Minet for her patience, and sharp advice through my thesis. I thank Professor Diego Dujovne from University of Diego Portales in Santiago, Chile, for hosting me in his laboratory as well as for his support and collaboration in the PEACH project, together with Gustavo Mercado, Ana Laura Diedrichs and Carlos Taffernaberry from the National University of Technology of Mendoza, Argentina. I thank Professor Steven D. Glaser from UC Berkeley for hosting me in his laboratory and allowing me to participate in real-world deployments in the Sierra Nevada together with Dr. Carlos Oroza, Sami Malek and Dr. Zeshi Zhang. I thank Mrs. Sofie Pollin, assistant professor at KU Leuven, Belgium and Mr. Fabrice Théoleyre, CNRS Researcher at the University of Strasbourg, France, for having accepted to review my thesis. I thank Lance Doherty, IoT Systems Architect at Analog Devices, and Mr. Marcelo Dias de Amorim, CNRS Researcher at the University Pierre et Marie Curie (Paris 6) for serving as examiners of my thesis. I thank Jonathan Muñoz, Dr. Tengfei Chang, Yasuyuki Tanaka, Dr. Mališa Vučinić, Dr. Ziran Zhang, and Dr. Rémy Léone for all their advises and support. I thank the undergraduate students I have had the privilege of working with for their help, and especially, Marcelo Ferreira, Felipe Moran and Fabian Rincon for their true engineering skills.

# Chapter 1

# Introduction

**In this chapter, we introduce the notions about the Internet of Things (IoT) that are used throughout this thesis. We explain the reasons that led us to carry out this work, and we present our contributions.**

## 1.1 Preliminaries

### 1.1.1 The Internet of Things

The Internet of Things (IoT) has so many definitions that the first thing I want to do is to rename it to fit my understanding. The broadest definition I have in mind is "tiny computers connected to the Internet". The term *Internet of Things* was first coined by Kevin Ashton in 1999 to describe the communication between a Radio Frequency Identifiers (RFID) device and the Internet. The term was rapidly adopted by governments and leading Information Technology (IT) companies to define a technology concept that would bring sustainability and economic growth. The way those "tiny computers" are connected to the Internet and what protocols they use varies a lot. The first "connected Thing" was a toaster, built in 1990 by John Romkey and Simon Hackett. The toaster was using TCP/IP and SNMP and was mains-powered. We now see devices the size of a grain of rice, battery powered, and that can communicate wirelessly **mesri16design**

In this document, I use the term *Industrial Internet of Things* (IIoT) to describe a subset of the IoT targeted towards reliability. The IoT brings many advantages such as ease of deployment and cost reduction, and its adoption by the Industry is increasing. The IoT is now a major component of the Industry 4.0, a vision where machines are capable of sensing their physical environment, taking decentralized decisions and collaborate with other systems

or users.

### 1.1.2 Wireless Sensor Networks

The term "IoT" encompasses many technologies. In this thesis, I focus on a subpart of the IoT that is the Wireless Sensor Networks (WSNs). Wireless Sensor Networks are networks of small electronic devices – called nodes – that communicate wirelessly using radio waves[1]. In Wireless Sensor Networks, nodes are organized around one (or multiple) collecting device(s) – called *sink* or *gateway*. The networks are usually "multi-hop", that is, nodes serve as data relays to forward data to the sink.

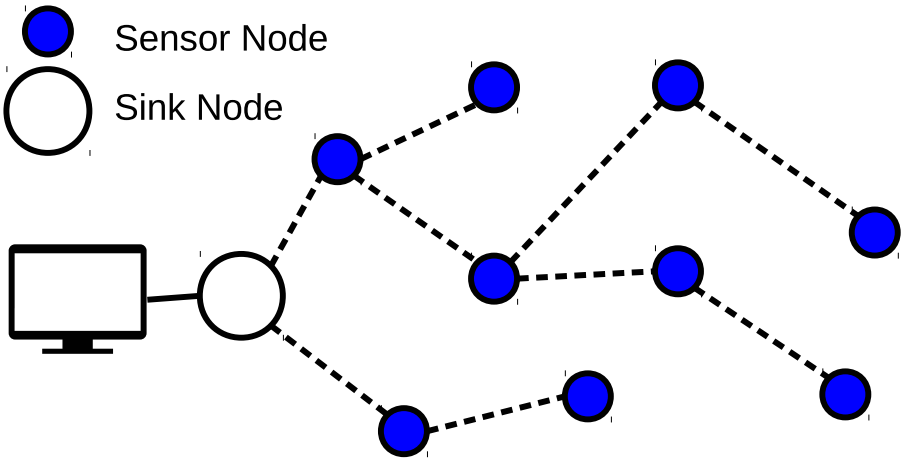



Figure 1.1: An example of multi-hop Wireless Sensor Network topology.

Wireless Sensor Networks (WSNs) need to offer at least the same capabilities as their wired counterpart. They need to be reliable, scalable, and durable, but also be easy to deploy, cost-effective and flexible. Most of those goals are achieved today. Solutions exist offering wire-like reliability with networks of hundreds of nodes, able to operate for years, deployable in only a few hours at a very low cost. The following chapter gives the reader an idea of how those requirements are achieved.

The major difference between traditional wireless networks such as IEEE802.11 (WiFi) or IEEE802.15.1 (Bluetooth) is that WSN nodes need to run over years on battery. This means that WSN nodes are heavily optimized to consume as little as possible. The main energy consumption source of those devices is the radio chip. Reducing the use of this chip reduces the node's energy consumption, yet it makes communication between nodes more complex.

[1] Electromagnetic waves on the radio spectrum (3 Hz to 3 THz)

Wireless is unreliable by nature. Unlike wired connection, wireless suffers from external interference, and is affected by different types of fading. This does not mean that there is no way of building a reliable service on top of it. There are multiple techniques available to mitigate the radio medium uncertainty; we survey those in Chapter 2. Wireless constraints are present under different forms, we now present them in more details.

### 1.1.3 The Wireless Impairment

We divide the wireless constraints in two parts, **external interference** and **fading**.

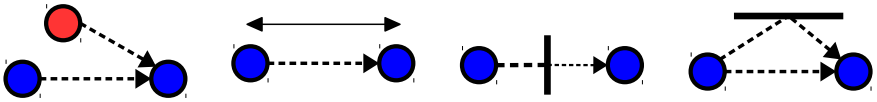
External interference occurs when multiple devices transmit at the same time on the same frequency[2]. In that case, the radio waves might collide, resulting in data loss. Such external interference is very present in the environments we study in this work as the technology we use communicates on an Industrial, Scientific and Medical (ISM) radio band. ISM bands are portions of the radio spectrum that are not subject to license. ISM bands are subject to rules (e.g. transmission power, data-rate) but do not require users to pay a usage fee. Regulations and bandwidth only varies a little across continents making their adoption by communication technologies easier. As a consequence, the radio communication contention is high in those bands, and particular care needs to be taken when designing protocols. We will see how this is achieved in Chapter 2.

The other wireless constraint is fading. Fading is the attenuation or variation of a radio signal. The first type of fading we can think about is called *Free Space Path-Loss* (FSPL). Free Space Path-Loss is the attenuation of a signal over distance. The radio energy dissipates making the amount of energy received lower than the amount of energy emitted.

The second type of fading I want to present is called *Shadowing* or *Shadow Fading*. Shadowing is also an attenuation but this time due to obstacles. When the radio wave hits an obstacle (e.g. wall, tree, water), the amount of energy that passes through is inferior to the amount energy before the obstacle.

The last type of fading I want to mention is named *Multipath Fading* and is less intuitive. Multipath Fading can be seen as self interference. When a signal is transmitted using radio waves, a receiver hears the signal that traveled following the direct line-of-sight (LOS), but also the echoes of the signal that bounced on nearby objects. Those "echoes" can be constructive or destructive. In the latter case, a fading occurs and the signal can become unexploitable. This phenomenon is called Multipath Fading. Multipath Fading is not related to distance, as shown in **watteyne09reliability**, **watteyne10mitigating** Multipath Fading can occur over very small distances. Multipath Fading increases

[2] This is not true for all technologies. For instance when using modulations such as OFDM.



(a) External Interferences (b) Free Space Path Loss (c) Shadowing (d) Multipath

Figure 1.2: The different Wireless Constraints we are faced with in WSN.

when the number of reflecting objects increases. Fortunately, the effects of Multipath Fading differs depending on the signal frequency. We will see how this is taken advantage of in Chapter 2.

Fig. 1.2 illustrates the main wireless constraints we are faced with when using WSNs. This list of constraints is not exhaustive, but rather gives an overview of the main challenges that need to be taken into account when designing WSN protocols.

### 1.1.4 Applications and Market Opportunities

Although computer-based monitoring has existed for years, the ability to be deployed densely at a very low cost opens a wide scope of applications to WSN and the IoT: Environmental Monitoring, Precision Agriculture, Building Automation, Factory Monitoring and many more. Dense monitoring information brings new insights and opens up a new era of competitiveness and growth. This list is not exhaustive and the details of each application type may not be true in every situation. This list is only presented here to give an overview of the potential IoT applications for IoT. Table 1.1 summarizes those types of applications and their requirements.

**Home Automation**: In those applications, the system can for instance control lighting, HVAC, or appliances in order to optimize a house consumption or comfort. To do so, the system is equipped with sensors and actuators that can communicate with each others forming networks. Those networks require low latency (e.g. switching light, changing music volume) and tolerate loss. Many IoT devices in home automation are powered with wires. Home Automation networks are usually in range of a gateway, meaning that the topology is a star.

**Smart Metering**: The goal of those applications is to periodically record consumption (e.g. electricity, gas) for monitoring and billing. Those applications require very low data rate and payload size (a few tenth of bytes per hours or per day) and do not require low latencies. Devices can be powered by batteries or with wires. They usually are located in urban environment at a range of a gateway (could be kilometers away).

**Environmental Monitoring**: The goal of these applications is to charac-

terize an environment in order to asses its health or understand its behavior. Unlike the other types of application, actuation is rarely involved. Devices are located far away from each others and sometimes not in range of the gateway. Unlike previously presented types of application, they might need to use other devices (relays) to forward their data to the gateway. They are usually located in harsh environments (subject to animals attack, tree falling, wind, rain, ...) in remote location. Replacing batteries or using wires is not an option in those use cases, those networks need to work during years on batteries. There is no strong requirement in reliability (e.g. a humidity data point loss is tolerated). As networks are located outdoor, they rarely suffer from multipath effects or external interferences.

**Industrial IoT (Industrial Monitoring and Automation)**: In those applications, the goal is to provide industrial machines the ability to sense their environment and react according to it. Those applications have very strong reliability requirements. When data is lost, it can directly impact the automation process (supply chain interruption) and result in loss of production. The devices are usually densely located, and surrounded by metal objects, thus multipath and external interferences are common in those environments.

| Application | Reliability | Latency | Topology | Links Dist. | Radio Env. | Power |
|---|---|---|---|---|---|---|
| Home Automation | Low | High | Star | $< 100m$ | Multipath & Ext. Int. | Battery or Wires |
| Smart Metering | Low | Low | Star | $> 500m$ | Multipath & Ext. Int. | Battery or Wires |
| Env. Monitoring | Medium | Low | Mesh | $> 500m$ | No Multipath, no Ext. Int. | Battery |
| Industrial IoT | High | Low | Mesh | $< 50m$ | Multipath & Ext. Int. | Battery |

Table 1.1: A summary of the IoT applications and their requirements.

We are now at a turning point for WSN and Industrial IoT. In the last ten years, the lack of a fully standardized network stack led companies to implement their own proprietary solutions. As a result, no interoperability was possible between the different IIoT companies, slowing down market adoption. This is over as Standards Developing Organizations (SDOs) are now proposing fully standardized, IPv6-ready, network stacks that provide the requirements the IIoT needs.

SDOs are professional associations that create and maintain normative technical documents (standards) that describe how network protocols should work. The standard elaboration process is open to the community and publicly available without any discriminatory conditions. This profits to vendors as they can penetrate market more easily – as those standard are usually worldwide adopted –, but also customers as they can access the technology without being tied to one company. The main SDOs in IIoT are the IEEE (link and physical layers), ETSI (complete machine-to-machine solutions), ISA (regulation for control systems) and the IETF (routing and network layers).

Thousand of proprietary IIoT networks are deployed in the world today. The interoperability between those networks and devices can widen the ap-

plication scope even more. While I can only foresee application-layer interoperability in industries, the coexistence of solutions in city-scale automation seems logical to me.

There is still work to be done to fully understand the capacity of IIoT networks. For instance, actuation and control loops is still at its infancy, and applications that provide industry-grade guarantees are in early development. The goal of my thesis is to reduce this gap, and trying to understand how much guarantee those networks can provide.

In this section, I defined what I mean by IoT and IIoT, what WSNs are and what main challenges to keep in mind when using WSNs. I show that applications can be very diverse and that the IoT market is still in expansion.

## 1.2 Contributions

My contribution is threefold:

- I participated in the deployment of four real-world WSN deployments and gathered more than 3M over two years, creating what we believe to be the largest real-world WSN dataset available.
- I analyzed, benchmarked and compared those datasets with IoT testbeds and defined a set of requirements that need to be taken into account when designing IoT protocols.
- I showed that determinism in Industrial IoT networks can be achieved and built a tool to estimate the performances of Industrial IoT networks.

### 1.2.1 Network Deployments and Data Collection

During the first year of my thesis I actively participated in the deployment of four real-world WSN deployments:

- **PEACH:** A frost prediction system for peach orchards in Mendoza (Argentina). We deployed 23 nodes to measure temperature and humidity at different locations in the orchard.
- **SnowHow:** A snow-pack monitoring system in the Sierra Nevada (California). We deployed 27 nodes to measure snow level, solar radiation and more in the mountain.
- **EvaLab:** A Smart Building monitoring system in Paris (France). We deployed 22 nodes to measure temperature in an office building.
- **SmartMarina:** A metering and management system for marinas in Agde (France). We deployed 19 nodes to measure electricity consumption in the marina.

Together, those deployments generated more than 32M sensor measurements and 3M network statistics. I built the rest of my work on top of those datasets.

### 1.2.2 Data Analysis and Comparisons

I first looked into the real-world network statistics and started my first network characterization and benchmarking study. I produced plots and results that indicate how well a network behaves, and found that in some cases links can be considered symmetric and that network churn (node parent change) could be only a few per day.

Then I compared real-world and testbed results. Using the IoT-LAB, a group of open testbeds in France, I collected dense radio connectivity traces (i.e. radio link qualities over time). Comparing those traces results with the ones collected in real-world deployments, I identified weaknesses of testbeds and provided a list of Key Performance Indicators (KPI) to watch for when designing IoT protocols.

### 1.2.3 Determinism in IIoT

As we said earlier, the radio medium is unreliable by nature. Given such statement is it possible to identify behaviors that are predictable in WSN ? By using Trace-Based Simulation to replay real-world and testbed results, I identified the limits of WSN regarding latency and network lifetime. I showed that when a set of conditions are met, we can predict the performances of a WSN. I designed and built a tool to help WSN protocol designers having a better idea of what to expect of their networks.

## 1.3 Thesis outline

**The second chapter** presents the state of the art. It starts from an overview of the existing WSN protocols before going into the details of the lastly adopted networking stack for the Industrial IoT.

**The third chapter** describes the methodology and assumptions we decided to take in this work. Unlike the mostly adopted approach that consists in reading the maximum number of papers and articles of the domain literature, we decided to start by manipulating networks in real-world conditions and understanding their real limits before proposing any improvement. I believe that this approach strengthened my opinions when taking improvement decisions and allowed me to easily spot unrealistic assumptions.

**The fourth chapter** details our real-world deployments. For each deployment we explain its application and challenges, the hardware and software choices we made, how many nodes were deployed, what data we collected.

We also give a first idea of the performance of the network and list the lessons learned.

**The fifth chapter** presents the experiments we ran in testbeds and the results we obtained by comparing testbeds with real-world deployments. In particular, I talk about the Mercator project, a dense radio connectivity data collection system. We identified the network Key Performance Indicators (KPI) and behaviors one must observe in order to validate an IoT protocol.

**The seventh chapter** focuses on studying the limits and trade-offs of WSN scheduling regarding latency and network lifetime. I explain why we think that Trace-Based Simulation is the right tool for such study and compare real-world results with theoretical limits. I finish by presenting the 6TiSCH Performance Estimator, a tool I built to estimate the performances of IIoT networks.

# Chapter 2

# State of the Art and Challenges

**In this chapter, we survey the related work in IoT, explain why we believe IoT networks can be deterministic, and expose the open issues and challenges.**

## 2.1 The IoT Standards

### 2.1.1 A Diversity of Applications

As IoT is a vast term, many protocols can be associated with it. In this section, I give a brief overview of the different protocols classified by category of applications presented in Chapter 1, namely Home Automation, Smart Metering, Environmental Monitoring, IIoT. This categorization might not be accurate for each type of application and is only used here to give the reader a first understanding of the variety of possible scenarios IoT can cover.

**Home Automation**: In Home Automation a small number of devices are deployed within a small range (e.g $< 15m$) to provide monitoring (temperature, humidity, noise...) and actuation (opening the windows, turning off the lights...). We only consider the devices communicating wirelessly, altough Home Automation also includes devices communicating using wires. The most broadly adopted wireless standards used in Home Automation are IEEE802.11 (WiFi) and IEEE802.15.1 (Bluetooth). WiFi-based networks consume a lot of energy (i.e. $> 10mA$ on average) and need to be powered with wires (or recharged frequently, as it is done with WiFi-enabled phones). Bluetooth based networks consume less energy, especially since the Bluetooth Low Energy (BLE) appearance, and targets applications that require a small amount of sensing points. Bluetooth devices can be powered by battery and can last a few years when using a low data rate. In its latest versions, Blue-

tooth can form mesh topologies but only with a limited number of devices. In this mode, the lifetime of the devices that relay information does go beyond a few weeks. Then comes 802.15.4-based networks, technologies such as Thread or Zigbee offer years of operation with devices powered by batteries. Those technologies can be densely deployed and cover larger areas than Bluetooth using a single gateway as they are meant to form a mesh topology. The last one I would like to mention is Z-Wave. Unlike the previously presented technologies that use the 2.4 GHz ISM frequency band, Z-Waves uses sub-GHz frequencies and thus can form links with longer range. Z-Wave can also form self-organizing mesh topologies to increase range and reliability. Again, this list is not exhaustive and many other technologies exist. I am only presenting the mostly adopted ones to illustrate the diversity of approaches.

**Smart Metering**: As said in Chapter 1, Smart Metering applications do not require low latency and high data-rate. Usually, only a few bytes are generated periodically (i.e. every hour) per meter. Technologies include Ultra Narrowband-based solutions like Sigfox, or Spread Spectrum-based solutions like LoRa. The Smart Utility Networks (SUN) task group is also working on a PHY amendment to address such applications. Note that Smart Metering can also be done using Power Line Carrier (PLC) but this is out of scope.

**Environmental Monitoring**: Those applications are very similar to Smart Metering as they are not producing a lot of data and rarely need to be densely deployed. As they are located in remote areas, they do not have a strong requirement in being able to cope with external interference or multi-path. Remote locations are however more likely to be to far for any cellular connectivity or power supply and thus require low energy consumption. They also need to be extremely solid and reliable as they need to work for years without human intervention in harsh condition (temperature, humidity, rain, wild life, ...).

**Industrial IoT**: Unlike previously presented applications, IIoT requires the combination of high reliability, years of battery lifetime, and be able to work even when densely deployed in radio environment where multipath and external interferences are present. This is the type of applications we focus on in this work. To the best of my knowledge, the only solution that can provide such guaranties today are TSCH-based technologies. We will define what it is and why we think it is the best suited solution available in Section 2.2.

### 2.1.2 Standardization and Interoperability

Many Standards Developing Organizations (SDOs) exist in the IoT world. The goal of these organizations is to design technical standards and protocols so that multiple technologies can talk the same language and thus, interoperate. The usual process is that private companies innovate and implement their own technology. As pioneers, they would lead this new technology for a few years and obtain recognition and market shares for that. Then other

companies will implement their own version of the technology and multiple similar solutions will coexist for a time. Finally, those companies, together with other technology experts, will merge efforts and create standards so that one technology gets adopted by all and that the different implementations can interoperate.

Standard and protocols are usually grouped into *abstraction layers* where each layer is responsible for a set of tasks (e.g. formating, compressing, routing, signaling). In networking, the complete set of layers is called a *Networking Stack* (or just *Stack*). The most widely used stack models are the Open Systems Interconnection (OSI) model and the Internet Protocol Suite, commonly known as the TCP/IP model. Each layer knows the tasks it has to do and what to expect from its upper and low layers. This layering distribution makes the protocols interchangeable. The main SDOs in IIoT are the Institute of Electrical and Electronics Engineers (IEEE), responsible for the lowest layers (close to the hardware), the Internet Engineering Task Force (IETF) responsible for routing and networking, the European Telecommunications Standards Institute (ETSI) famous for Machine-to-Machine (M2M) standards, and the International Society of Automation (ISA) oriented towards control system regulations.

A wide range of standards and protocols exist in the IoT world, but until recent year, no entire networking stack existed to answer the Industrial IoT requirements. Such a stack requires to be **IPv6-ready** to be able to communicate with other devices on the Internet, **Low Power** to operate during years on battery, and **Highly Reliable** to offer Quality of Service (QoS) guarantees.

One of the main idea behind the Internet of Things is that devices need to be addressable in order to interact with the rest of the Internet. Being able to communicate with other devices on the Internet opens the way to a wide range of applications. As an example, Internet allows to bypass the limits of geographical distances, two machines can exchange information across continents to optimize a delivery process. The Internet Protocol (IP) is the main addressing protocol on Internet. Its first version (IPv4) was released in 1981 and we are now moving towards the latest version, IPv6. In 2007, an IETF working group called “IPv6 over Low-Power Wireless Personal Area Networks” (6LoWPAN), was created to bring IP capabilities to low power and constrained devices **rfc4944** This resulted in the creation of the 6LoWPAN adaptation layer, a set of protocols and methods to enable efficient transport of IPv6 packets over IEEE802.15.4 frames. At that point, the **IPv6-ready** goal was achieved, but we were still lacking the reliability and low power consumption.

Right after, the IEEE task group ”4e” (TG4e) was created, chartered with defining an amendment to IEEE802.15.4 for the MAC layer to better support the industrial markets. Among other mechanisms, the TG4e chose to incorporate time slotting and channel hopping techniques embodied in a MAC layer mode called Time Slotted Channel Hopping (TSCH). In 2016, those

changes got merged into the standard's new version, IEEE802.15.4-2015.

This was, however, not enough to yield a fully standardized IIoT stack, as there was still no standard for the networking layer to access and reserve MAC resources. A component was missing to bridge the gap between the IEEE802.15.4 TSCH and the networking layer. The IETF 6TiSCH Working Group (WG) was created in 2013 to bridge that gap and propose a fully standardized stack for the Industrial IoT. At the time of writing, two Request for Comments (RFC) were published in the working group **rfc7554**, **rfc8180** and the 6TiSCH stack architecture is about to be published **draft-ietf-6tisch-architecture**

In the next sections we explain the concepts and mechanisms of TSCH and detail all the layers of the 6TiSCH stack, the Industrial IoT networking stack.

## 2.2 Time Slotted Channel Hopping

Time Slotted Channel Hopping (TSCH) is a channel access method for shared medium networks. TSCH is designed for applications that require reliability and ultra long battery life.

### 2.2.1 History & Description

In 2006, a startup company called *Dust Networks* introduces TSMP (Time Synchronized Mesh Protocol) **pister08tsmp** a protocol for self-organizing Wireless Sensor Networks (WSN). In a TSMP network, the devices – called motes, as small particles of dust – are synchronized to each other and communicate respecting a time schedule. Time is divided into slots (timeslots), similar to other Time Division Multiplexing (TDM) systems. The devices know when to sleep, transmit or receive, and stay asleep most of the time, allowing extremely low energy consumption. On top of the time scheduling, TSMP devices use *Channel Hopping*, a technique in which transmissions are distributed over time and radio channels. Hopping through channels increases the communication reliability over noisy environment.

In 2008, the base concepts of Dust Networks' TSMP got included into two low-power wireless industrial standards, WirelessHART (2008) and ISA100.11a (2009) under the name *Time Synchronized Channel Hopping*. These standards have been very successfully rolled out in the industrial market (industrial process monitoring, factory automation). WirelessHART is an interoperable wireless standard designed to provide reliable, cost-effective, high-quality system for industrial wireless sensing applications **chen10wirelesshart** WirelessHART got widely adopted as it was

backward-compatible to the legacy (wired) HART[1].

In the mean time, the IEEE 802.15 Task Group 4e was chartered to define a MAC amendment to the existing standard 802.15.4-2006 to "better support the industrial markets". They chose to reuse the TSMP concepts and named it *Time Slotted Channel Hopping* (TSCH). The amendment was published in April 2012 and got incorporated into the IEEE802.15.4-2015 standard **std_ieee802154-2015**

The 6TiSCH working group **palattella156tisch** is now standardizing the use of IPv6 on top of IEEE802.15.4-2015 TSCH technology.

### 2.2.2 A Slotted Structure

In TSCH, time is divided into **time slots** (also called *slots*) that typically last around 10 ms. Time slots are grouped into a **slotframe** that repeats over time (as depicted in Fig. 2.1). The way time slots are organized into slotframes is called a *schedule*. To each time slot is associated an action: *sleep*, *transmit*, or *receive* so that each node knows in advance what to do next. If the action is *sleep*, the node turns its radio off and waits for the duration of a time slot. When sleeping, the node only consumes a few $\mu A$ at 3.6 V. If the action is *transmit*, the node turns its radio on and transmits a frame. If the frame requires to be acknowledged, the node then listens for an acknowledgement frame in the same slot. If the action is *receive*, the node turns its radio on and listens for a frame. If the frame requires to be acknowledged, it transmits an acknowledgement. A schedule example is depicted in Fig. 2.2. In the first time slot, node `D` transmits to node `B` and node `C` transmits to node `A`. In the second time slot (about 10 ms later) node `B` transmits to node `A`. We will see in Section 2.3 how the schedule is managed.

As depicted in Fig. 2.4, slot are long enough (e.g. 10 ms) for both the transmission of a data frame and the transmission of an acknowledgement (ACK) frame. Unlike with wired transmissions, there is no way to know if a collision occurred during a transmission when using wireless communication. The only way to know if a transmission failed or succeed is to ask the receiver for confirmation. This is done using an acknowledgement (ACK) frame. When the first sender receives an ACK, it knows the receiver got the message. Of course, there is no way for the receiver to know if the first sender correctly received the ACK. Rather than sending a second ACK, nodes usually assume that if the first message went through, the probability of success of the ACK message is high (as the delay between the two messages is short and the frequency does not change).

TSCH defines two types of cells:

- A **Shared Cell** is a cell in which multiple communications can occur.

[1] Highway Addressable Remote Transducer (HART)

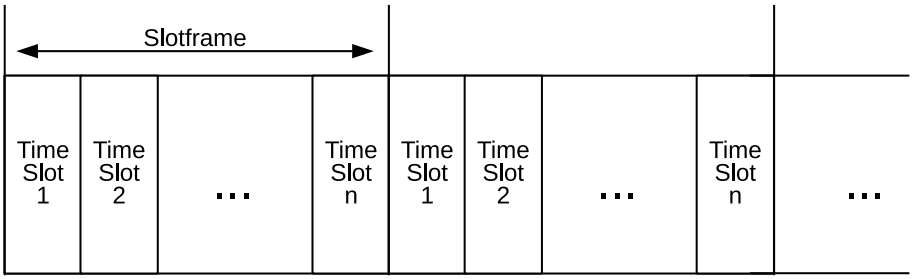



Figure 2.1: Time Slotted Channel Hopping slotframe.

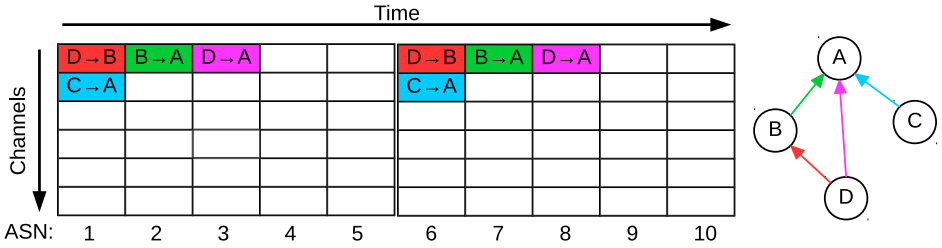



Figure 2.2: Time Slotted Channel Hopping schedule. A minimal schedule example.

When using shared cells, node need to expect collisions and thus implement mechanisms to prevent them.

- A **Dedicated Cell** is a contention-free cell. The cell is dedicated to a pair of nodes, in one direction so it has a fixed source and a destination. When using a dedicated cell, a source node can assume that no other transmitter will use that same dedicated cell with the destination node.

The main source of the nodes' energy consumption is the radio; there is a direct link between the number of cells that are scheduled and the energy consumed. Typically, nodes stay in *sleep* mode most of the time allowing them to consume only a few tenth of $\mu A$ at 3.6 V on average, and achieve years of battery lifetime. Xavier Vilajosana *et al.* **vilajosana14realistic** propose a realistic model to estimate a node's energy consumption based on the number and type of cells that are used in the schedule. We use those results in Chapter 6 to evaluate the relation between energy consumption and latency in different environments and network configurations.

On top of time division, TSCH also does frequency division. The available frequency band is divided into *channels* by the physical layer. Each channel is an available resource the MAC layer can use. Multiple channels can be used at the same time, meaning that in each time slot, all the available channels can

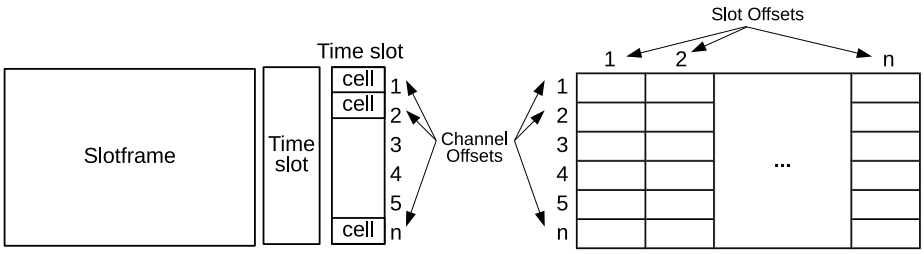



Figure 2.3: Time Slotted Channel Hopping Taxonomy.

be used in parallel. This results in the notion of a **cell**, where a cell is a part of the schedule that can be identified with a *slot offset* and a *channel offset* (as depicted in Fig. 2.3).

Each channel is not equally affected by external interferences or multipath fading, thus using multiple channels augments the radio environment diversity and reduces the probability of failure **watteyne09reliability** For instance, if the third channel is used continuously by another technology (e.g. WiFi) and all other channels are free, using all the channels available will give a probability of failure of $1/number\ of\ channels$. To make sure all channels are used and well distributed, TSCH uses (2.1).

$$frequency = (ASN + channel\_offset)\%number\_channels \tag{2.1}$$

Where, $channel\_offset$ is the channel offset of that cell and $number\_channels$ is the total number of channels available. The Absolute Sequence Number (ASN) is a global time slot counter allowing all nodes to share the same sense of time. The ASN value increases at every slot: two consecutive cells on the same channel offset will not use the same channel, and that a same cell will not use the same channel in two consecutive slotframes (unless the slotframe length and the number of channels are not mutually prime). At every time slot the channel changes, this is called "channel hopping". The resulting frequency diversity greatly increases reliability and stability of the links **watteyne09reliability** making TSCH a perfectly suited MAC layer technique for a reliable IoT.

### 2.2.3 Time Synchronization

Because every communication in the network is scheduled, node must stay synchronized. Nodes locally keep track of time with an internal clock. This is typically done using a crystal oscillator (although on-chip ring oscillators were proven to work for millimeter-scale motes **wheeler17crystal**). Those clocks are never perfectly accurate and drift in time with respect to one another. Nodes therefore need to periodically resynchronize.

Clock drift is usually measured in *parts per million* (ppm), that is how many clock "ticks" are off in a million. A typical crystal drift in WSN devices is between 10 ppm and 50 ppm. A 10 ppm clock drift corresponds to a maximum drift of 10 $\mu$s per second or $+-0.864s$ per day. Although this number might appear small, it makes a difference as a slot duration is usually in the order of magnitude of 10 ms. This is even more important as the clocks of two neighbor nodes might drift in opposite direction (one going 10 ppm slower, the other 10 ppm faster) making the nodes' relative clock drift twice the absolute clock drift.

What is the maximum desynchronization two nodes can tolerate and how nodes can resynchronize? Fig. 2.4 depicts the different steps performed within a cell when a transmission occurs. The transmission starts exactly after a fixed delay (`TxOffset`). If the destination node (DST in Fig. 2.4) starts receiving at the same `TxOffset`, it might miss the beginning of the frame as its own time view might not be the same as the transmission source (SRC in Fig. 2.4) one. Thus, the destination starts listening `GuardTime` before `TxOffset` and keeps listen until it receives a frame, for a maximum listening duration of `GuardTime`. If the relative desynchronization is higher than `GuardTime`, the transmission fails. Another way of reducing the impact of clock drift is to compensate for it using its estimated drift. Quartz manufacturers provide a drift estimation depending on temperature. To take this into account, WSN hardware typically comes with a temperature sensor so that nodes can compensate the clock drift in software. Adding a `GuardTime` and compensating for temperature helps reduce the impact of clock drift until some limits.

The nodes needs to resynchronize periodically. In TSCH, resynchronization is done by exchanging frames. Each IEEE802.15.4 frame is timestamped: when a node receives a frame, it calculates the delta between the reception time and the frame timestamp and can recalibrate its clock. How often a node's clock needs to be recalibrated depends on the drift and `GuardTime` values. With a 30 ppm drift and a 1 ms `GuardTime` (typical values), two nodes need to resynchronize every $1ms/30ppm = 33s$. If no data frame is sent during that period, nodes exchange dedicated *keep-alive* frames for resynchronization. Considering a transmission of 4 ms (typical duration) every 33 s would lead to a $4/33 = 0.12\%$ duty cycle. The cost of resynchronization in TSCH is hence very low.

In the next section, we describe the different protocols and mechanisms that form what we call an Industrial IoT Stack. We give an overview of the different approaches taken by the research community and companies to result in a fully standardized stack for the Industrial IoT: the 6TiSCH stack.

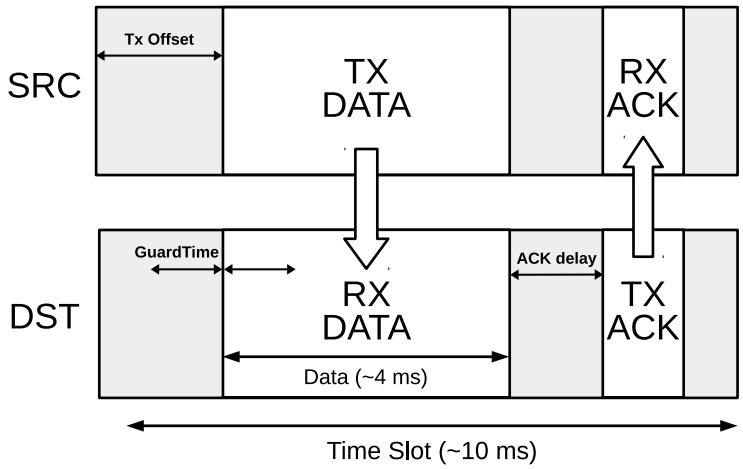



Figure 2.4: A Time Slot Timing Example

## 2.3 Industrial IoT Stack

A networking stack is the assembly of a series of network protocols. It is called a *stack* as those protocols can be grouped in layers that are piled up from the higher level (i.e. the application) to the lower layer (i.e. physical). In this section, we present a subset of the different protocols that exist in the IoT, and describe in more details the one that got selected to form 6TiSCH, the networking stack for the IIoT.

### 2.3.1 The 6TiSCH Networking Stack

6TiSCH is the name of the Internet Engineering Task Force (IETF) working group in charge of designing a networking stack for the Industrial IoT.

The goal of the 6TiSCH working group is to provide a standardized networking stack for the IoT. The stack needs to be **Low Power**, **Highly Reliable** and **Internet-Enabled**. The 6TiSCH name comes from the combination of IPv6 and the TSCH mode of IEEE 802.15.4, two proven technologies selected as they provide interoperability and reliability respectively.

The proposed stack is flexible as there is room to fine-tune several protocols in the stack to a particular application. The goal is to propose a reference stack that Industrial IoT designers can adapt to their particular needs. As a proof of concept (PoC), an RFC was published to define a 6TiSCH minimal mode of operation that 6TiSCH-compliant devices should implement **rfc8180**

The 6TiSCH stack is similar to the Open Systems Interconnection (OSI) model or TCP/IP layering model. The protocols are separated into abstrac-

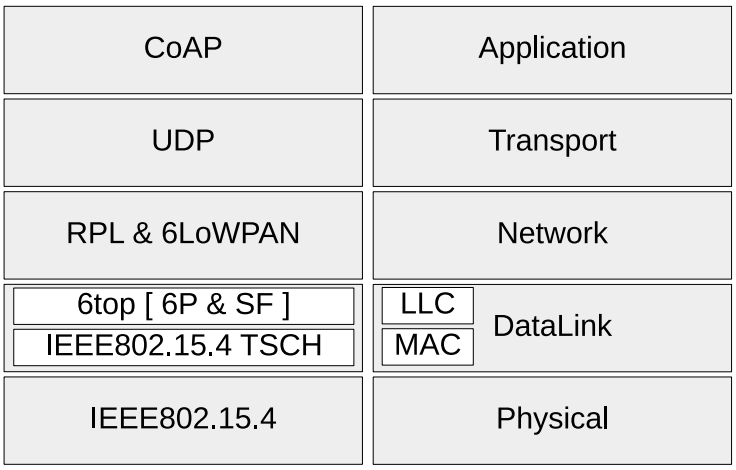



Figure 2.5: An overview of the 6TiSCH Networking Stack

tion layers, each layer only interacting with its upper and lower layer. By following this separation of work, a stack can easily be adapted to a particular use case, just by switching a protocol of one layer with a more suited one. The 6TiSCH stack is depicted in Fig. 2.5 and is separated in five layers that we will define in the following sections.

In the following subsections we describe, layer by layer, the main protocols and mechanisms that exist to build an IIoT stack and present which ones were selected to be part of the 6TiSCH stack and why.

### 2.3.2 Physical

Wireless nodes communicate by sending digital information as electromagnetic waves over the air. To transform a digital information into an electromagnetic waves, the digital information must first be translated into an analog signal. This transformation is called *modulation*. Then the analog signal needs to be amplified and sent over the antenna. A typical low-power radio outputs 0 dBm (1 mW). As explained in Chapter 1, the radio signal power might fade due to obstacles, distance or external interferences. As a result, a receiving node needs to amplify the received signal before demodulating it. To extract relevant information from the signal, the ratio between the signal carrying the data and the ambient radio noise must be kept high. This is quantified using the Signal-to-Noise Ratio (SNR).

The amplifier and modulator components draw a significant amount of energy, making the radio the most power-hungry part in most designs. Those components do not consume energy when off. The challenge is thus to limit their usage while providing reliable communication. An energy-efficient communication stack usually uses those components less than 1% of the time.

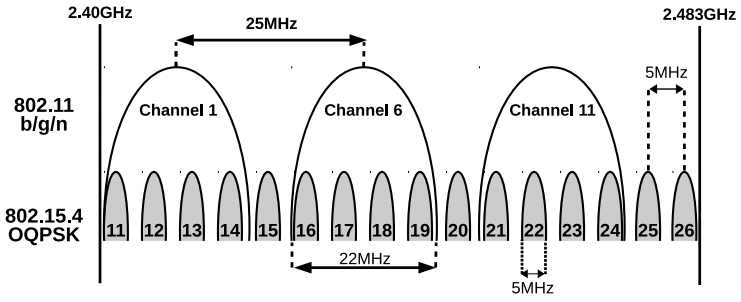



Figure 2.6: Overlapping Channels in 802.11b/g/n and 802.15.4 OQPSK

The PHY layer is responsible for the activation and deactivation of the radio, preamble detection (identifying symbols that correspond to a known modulation), and energy detection (ED). The chosen standard in 6TiSCH is IEEE802.15.4-PHY as it is the most prominent standard in low-power radio technologies. IEEE802.15.4-PHY proposes multiple modulation schemes. In 6TiSCH, the default modulation scheme is Offset Quadrature Phase-Shift Keying (OQPSK)**rfc8180** on the 2.4 GHz ISM band. In this modulation, the digital data in translated into an analog signal by modulating (changing) the signal phase. OQPSK is a variant of phase-shift keying modulation using four different values of the phase to transmit. This modulation technique is also used in Bluetooth and RFID. IEEE802.15.4 with OQPSK provides 16 independent channels (numbered from 11 to 26), that is, sending on channel 12 will not impact another communication (using the same technology) on channel 11 or 13. Fig. 2.6 depicts the channel overlapping between 802.15.4 OQPSK and 802.11b/g/n (WiFi).

### 2.3.3 DataLink

In IEEE802, the DataLink layer in divided into two sub-layers: Medium Access Control (MAC) and Logical Link Control (LLC). The MAC sub-layer is responsible for controlling how devices gain access to a physical medium (e.g. which channel to use). The LLC sub-layer is responsible for bridging the gap between the MAC sub-layer and the network layer, and also controls error checking and frame synchronization. 6TiSCH uses IEEE802.15.4-2015 with the TSCH mode at the MAC layer and the 6TiSCH Operation Sublayer (6top) for LLC. The 6top sub-layer includes the 6top Protocol (6P) that defines the commands and interaction between nodes to reserve resources (cells) and the Scheduling Function (SF) that is internal to each node and defines when to add or delete cells in the schedule. Work is being proposed at the IEEE (802.15.12 PAR) for an LLC that would logically include the 6top sublayer.

### MAC

The Medium Access Control (MAC) sub-layer provides an interface between the physical layer (PHY) and the LLC layer. The different MAC designs can be separated into two different paradigms, *contention-based* and *time-divided*.

In contention-based MAC, nodes need to make sure the radio medium is not used by other transmissions before transmitting. This is usually done using Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA), a method in which nodes send a short message named Request to Send (RTS) before sending, to make sure no other node is transmitting. If the receiver has no ongoing transmission, it replies with another short message named Clear to Send (CTS). Then the actual data transmission can start. This technique is used in broadly adopted technologies such as WiFi. Contention-based MAC protocols are appealing as they allow multiple devices to share the same radio channel without being pre-coordinated.

One of the first adaptation of existing contention-based MAC protocols for Wireless Sensor Networks is SMAC **ye02energy** S-MAC is designed to reduce energy consumption, while supporting good scalability and collision avoidance. S-MAC consumes from 2 to 6 times less energy than traditional IEEE802.11 MAC protocols. Still, they show a 30% duty cycle, that is way above what is expected for a solution to run during years. Contention-base solution cannot provide strict guarantees with ultra low-power consumption.

The other MAC paradigm is *time-division*, where devices share the same radio frequency by using it at different times. This is known as Time-Division Multiple Access (TDMA). Time is usually divided into slots of the same duration. As opposed to contention-based solutions, nodes need to agree on a time-division scheme (e.g the duration of a time slot). TDMA can be coupled with Frequency-Division Multiple Access (FDMA) in which a frequency band in sub-divided into *channels* and multiple channels can be used at the same time without interfering with each others. This is now a largely adopted technique in Wireless Sensor Networks[2].

Before 2012, the IEEE802.15.4 MAC sub-layer was designed for star networks, in which all motes communicate directly with a central coordinator mote. The way the sub-layer was built would not match IIoT use cases for two reasons:

- **Link Reliability**. As described in Chapter 1, the radio environment is unreliable in nature. The quality of a transmission over one frequency can change from one second to another. Until 2012, the MAC sub-layer was not using Channel Hopping.

[2]At the time of writing more than 70k networks are deployed in the world using the SmartMesh IP solution that include such technique. This technique is also included in WirelessHart, a protocol used in many industries for factory automation.

- **Relay Energy Consumption**. To have a mesh network, some nodes need to act as relays (routers). Until 2012, relaying nodes needed to keep their radio on all the time (100% duty cycle).

In 6TiSCH, the chosen MAC layer is the IEEE802.15.4-2015 with the TSCH mode. Using the TSCH mode allows both low duty cycle (<1%) and high reliability (>99.999%) **doherty07channel** IEEE802.15.4-2015 describes the way TSCH works and its typology, but does not define how the schedule is built.

### LLC

The way cells are organized in time and channels is called the schedule. The management of the schedule is a crucial task as it has a direct impact on latency, reliability and energy consumption. The number of allocated cells in TSCH is directly related to bandwidth. The more cells a node has, the more opportunities to transmit or receive. Scheduling with TSCH can be done following five paradigms: Static Scheduling, Neighbor-to-Neighbor Scheduling, Remote Scheduling, Hop-by-hop Scheduling, and Id-based Scheduling.

In **Static Scheduling**, a fixed schedule is defined for the entire network, no matter the bandwidth required by each node, no matter the number of nodes. It means that multiple transmissions can occur in the same cell, resulting in potential interference and data loss. Static Scheduling is often used to bootstrap the network or as a fall-back mode (when other scheduling techniques failed to operate).

In **Neighbor-to-neighbor** Scheduling, nodes negotiate cells in a distributed manner. Each node exchanges messages with its neighbors to allocate/deallocate cells between each other.

In **Remote Scheduling**, a central entity (e.g. the gateway) computes a schedule for each mote. The advantage of that approach is that one entity has a global view of the network and can thus take decisions that can optimize the overall performances. The drawback is that it takes more time and network usage, as each node needs to send information to the scheduling entity. The central entity might then have information that is not up to date.

In **Hop-by-hop Scheduling**, nodes can reserve cells among a path to a destination. In 6TiSCH this is called a *Track*. A Track is the 6TiSCH instantiation of the concept of a *Deterministic Path* **ietf-detnet-architecture-07**

In **Id-based Scheduling**, nodes decide which cells to allocate depending on a unique identifier (MAC address or other). If this identifier is obtained from the sender, it is called Sender-based Scheduling, and if obtained from the receiver, it is called Receiver-based scheduling. When two nodes want to communicate and know their identifiers, they choose one of the two identifiers and translate it into cell coordinates (i.e. slot offset and channel offset). How to translate an unique identifier into cell coordinates is not defined. The only

constraint is to make sure the mapping to a cell coordinate is unique. Cells allocated using Id-based scheduling are shared. This approach is presented in Orchestra **duquennoy15orchestra** and is now being merged in the 6TiSCH Minimal Scheduling Function (MSF). This technique is efficient as it does not require any wireless communication and negotiation.

While the standard does not define how the schedule is built and operated, scheduling has major impacts on the network performance. In their survey **hermeto17scheduling** Teles Hermeto *et al.* provide an extensive description of the different scheduling techniques available. They classify the scheduling techniques according to their paradigms (i.e. centralized or distributed) and optimization goal (e.g. low latency or reliability). They point out that centralized schedules are ideal for static topologies with periodic traffic and distributed scheduling is more suited for mobile topologies and traffic that is not determined in advance.

As we mainly use centralized scheduling in this thesis, I will now describe the corresponding state of the art. One of the pioneer work in centralized scheduling algorithm over TSCH proposed by the research community is the Traffic Aware Scheduling Algorithm (TASA) **palattella12traffic** In TASA, the schedule is managed by a centralized entity named the Path Computation Element (PCE). TASA is built for periodic convergecast traffic (the entire traffic is addressed to the gateway) and works over DODAG where node have only one parent. TASA aims at building compact schedules, that is, minimizing the offset of the last cell in the schedule. It starts to allocate bandwidth (cells) to the most constrained nodes (the nodes that carry the most traffic). TASA then uses matching and coloring heuristics to find the smallest schedule taking into account the traffic and the topology. TASA guarantees optimal schedule compactness but does not provide any guarantees in terms of reliability, mainly because it assume perfect links and no retransmissions.

Gaillard *et al.* **gaillard16high** proposed an extension of TASA ($TASA_RTX$) to take into account retransmissions and fragmented packets. They build a schedule that complies with reliability expectation by adding extra cells that are used in case of consecutive retransmissions. They take into account link quality, packet fragmentation and traffic changes.

The same authors further extend their research by proposing Kausa, a KPI-aware scheduling function **gaillard16kausa** The authors consider that multiple applications use the same network, and that each application has its own set of requirement and traffic flow. They design Kausa, a centralized scheduling algorithm that builds resource paths that guaranty per flow QoS.

Khoufi *et al.* **khoufi17scheduling** use centralized scheduling to build a schedule that is compliant with both latency and reliability requirements. To do so, they introduce the concept of debt-based scheduling, that is a mechanism that schedules first the node with the highest debt. They define the debt of a node based on the amount of traffic it needs to carry out and its depth in the network (the number of hops to DAG root). We reuse this concept in

Chapter 6.

In 6TiSCH, the LLC sublayer is embodied by the 6TiSCH Operation Sublayer (6top). 6top provides a management interface that enables an external management entity to schedule cells and slotframes named the 6top Protocol (6P) as well as a structure and formalism for the scheduling mechanisms called the Scheduling Function (SF). 6top defines two types of cells: **Hard cells**, cells that cannot be modified (i.e read-only), and **Soft cells**, cells that can be modified.

The scheduling tasks are done by the Scheduling Function (SF) components. A node may support multiple SFs at the same time, each SF is identified with an SFID. 6TiSCH only defines a set of requirements a SF must have (e.g. the SFID) but does not define how the SF computes the schedule.

As an example, a scheduling function named Minimal Scheduling Function (MSF) is proposed in RFC8180 **rfc8180** defined in **ietf-6tisch-msf-00** and described in **chang15adaptive** MSF defines both the behavior of a node when joining the network, and how the schedule is managed in a distributed manner. MSF uses a combination of Neighbor-to-neighbor Scheduling and Id-based scheduling. During the join process, MSF defines a set of steps a node follows to allocate the minimal set of cells its will use to start communicating with the network. After the joining process, MSF dynamically modifies the schedule to continuously adapt to the application and routing changes as well as to handle the schedule collisions.

The Scheduling Functions (SFs) decide which cells to allocate/deallocate locally, but does not apply those changes alone. To do so it uses the 6top Protocol (6P). The 6top Protocol (6P) defines the commands nodes send to each others to add, delete or relocate cells (redefine the location of the cell in the schedule) **draft-ietf-6tisch-6top-protocol** When the SF takes a scheduling decision, it triggers a 6P mechanism that is called a "6P Transaction". A "6P Transaction" is a series of messages a node and its neighbor exchange to negotiate the modification of their schedules. An example of a 2-step 6P Transaction is depicted in Fig. 2.7 After a 6P transaction, if both node agree, they modify their schedule. It is important that the two nodes keep their schedule consistent. If a node `A` has a transmit cell and node `B` does not have the corresponding reception cell, this results in communication failure.

### 2.3.4 Network

The Internet Protocol version 6 (IPv6) is the latest version of the Internet Protocol (IP) that provides devices addressing across the Internet. Devices are often considered part of the IoT domain when they are able to use IP. Having IPv6 capabilities simplifies the integration of a technology into a production system.

The IETF "IPv6 over Low-Power Wireless Personal Area Networks" (6LoWPAN) working group was created in 2005 to provide an adaptation

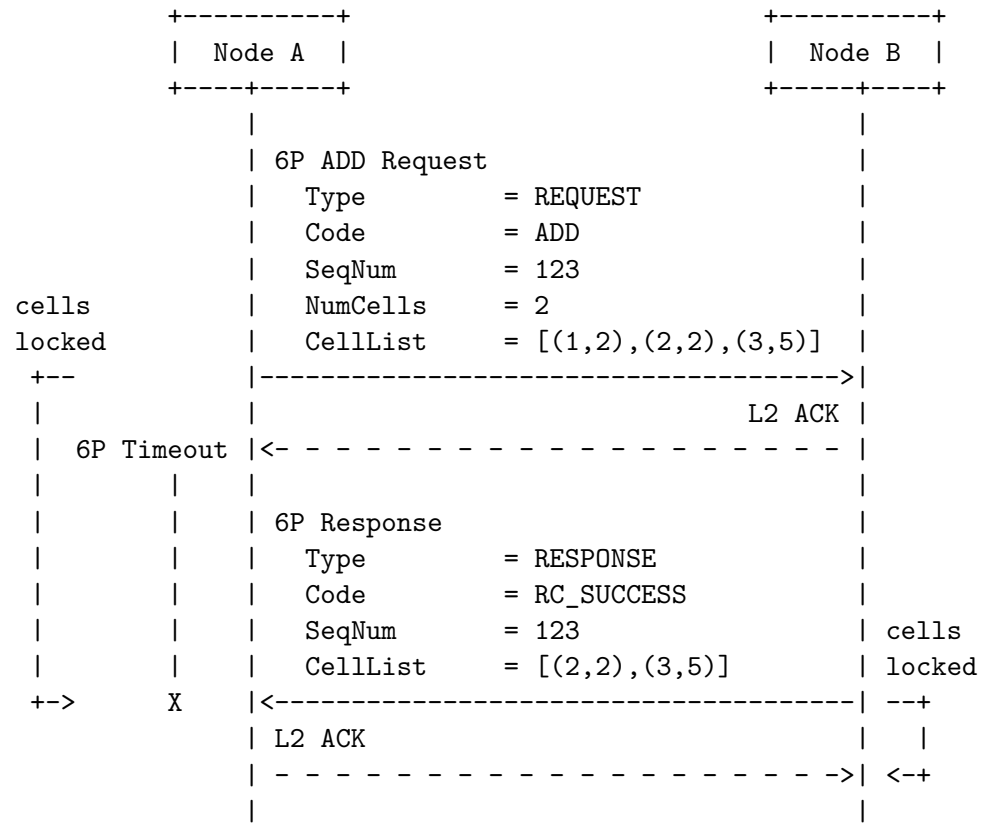



Figure 2.7: A 2-step 6P Transaction (borrowed from **draft-ietf-6tisch-6top-protocol**).

layer for IPv6 to work over IEEE 802.15.4 and resulted in a set of mechanisms known as *6LoWPAN* **rfc4944** Adapting IPv6 to constrained device networks is not straightforward and the main obstacle is size. The IPv6 Maximum Transmission Unit (MTU) is 1280 B and the IEEE 802.15.4 maximum frame, size is 127 B. 6LoWPAN thus defines mechanisms to compress and fragments datagrams.

6LoWPAN allows IPv6 to function on top of IEEE 802.15.4, but does not define how neighbors are selected and how the network is formed. In 6TiSCH, the default routing protocol is the "Routing Protocol for Low-Power and Lossy Networks" (RPL) **rfc6550** RPL supports a wide variety of datalink layers and is not only designed for WSN. RPL can build up network routes, adapt to a changing topology and distribute routing knowledge among nodes. RPL forms a Destination-Oriented Directed Acyclic Graph (DODAG), that is a multi-hop routing graph rooted at a central point (named gateway, root or sink). The routing graph is usually created by accounting for link quality and nodes attributes, based on a gradient-based approach. Which parameters to take into account is defined in an *Objective Function* (OF). The OF chooses the parameters to use based on the routing objective (e.g. create short paths, build redundant paths). Each node thus obtains a *Rank* that denotes its virtual distance to the DODAG root.

The DODAG construction starts at the root. The root periodically broadcasts a control message named DODAG Information Object (DIO). A DIO contains the rank of the node that sent it, as well other routing configuration parameters. When a joining node receives a DIO, it adds the DIO transmitter node to its list of potential parents. After receiving a few DIOs (the number of DIOs to wait for is defined in the OF), the joining node selects one of the potential parent and defines it as its *preferred parent*. How to select the preferred parent is also defined in the OF. For instance, a node can select its parent given its *Rank*, or given the *Rank* that will provide to the joining node. Once the joining node has selected its preferred parent, it redirects all its packets addressed to the sink to that parent. The joining node then computes its own *Rank* and starts broadcasting its own DIOs. Fig. 2.8 depicts the steps of a DODAG construction using hop count as the routing metric.

Once the routing and scheduling are settled, the application can start.

### 2.3.5 Application and Transport

As the goal is for IoT devices to be able to talk with the rest of the Internet, IPv6 is not sufficient. IPv6 provides a way to address other devices, but does not specify the type of data it carries or how this data is formatted. This is the task of the Application layer. The most famous Application layer protocol is the HyperText Transfer Protocol (HTTP). Two applications that use HTTP are able to communicate as they know in advance the format of the data they will manipulate, and what the communication steps are they need to respect

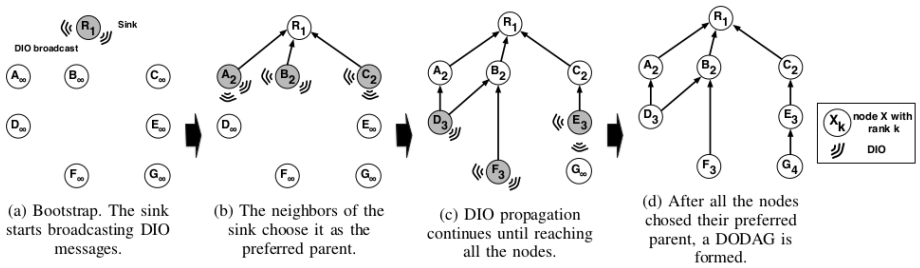



Figure 2.8: DODAG construction with hop count as a routing metric. Figure taken from "The Love-Hate Relationship between IEEE802.15.4 and RPL" **iova17love**

for the conversation to be carried out well. Classical wired networks do not have tight restrictions in terms of packet size or energy consumption, and are thus not optimized for minimum packet overhead. To be able to interoperate with such networks, the 6TiSCH stack needs to have techniques to compress the communications without compromising its content.

The Constrained Application Protocol (CoAP) over User Datagram Protocol (UDP) is the default choice for Application and Transport Layers in 6TiSCH. CoAP can be seen as a compression of HTTP with extra features built toward IoT such as IP multicast for group communication. CoAP integrates well with IPv6 and 6LoWPAN that are described in Section 2.3.4.

In this section, we saw that the 6TiSCH working group assembled what I believe to be the right protocols and mechanisms to build a stack for the Industrial Internet of Things that is **Highly Reliable**, **Low-Power** and **Internet-Enabled**. We will now see what the next steps are for this technology, and what the open issues and challenges are.

## 2.4 Open Issues and Challenges

### 2.4.1 Benchmarking IoT

To design and validate an IoT communication protocol, one needs to test its capabilities in different scenarios (e.g. number of nodes, position of the nodes, data rate, etc) and environments (indoor/outdoor, with or without external interferences, etc). There are three ways this can be carried out: simulation, testbed, and real-world. Unfortunately each solution has its drawbacks. Simulation is fast, flexible and does not requires hardware, but rarely represents the reality well. Testbeds better reflect reality, especially in terms of radio propagation and interferences, but are static in the sense that they represent only one radio environment. Real-world deployments present the same pros and cons than testbeds, with the advantage of being tied to an application. There is no need to make the network behave to fit one application as it is al-

ready designed for it (e.g. node positions, temperature and humidity changes, data rate, etc). Ideally, a protocol needs to be tested in as many scenarios and environments as possible. Unfortunately, validation is often done only with a subset of the possible application configuration **boano18iotbench** There is a need for a consistent way to test, validate and compare deployments and their results.

The IoT Benchmark Initiative (IoTBench) was recently created to provide a set of tools and best practices to enable fair comparison and repeatability of experimental results. They define 3 types of parameters and metrics an IoT benchmark should have: *Inputs* (e.g. number of nodes, data rate), *Outputs* (e.g. the nodes' energy consumption) and *Observed* (e.g. external interferences). I strongly believe this benchmarking initiative is the right approach for providing reliable solutions for the IoT.

During my work, I participated in the deployment of multiple real-world solutions and ran testbed experiments. In order to compare them, I adopted a way of formating "traces" (network statistics and radio connectivity) and extracted Key Performance Indicators (KPIs) that I present in Chapter 5. This trace format is generic enough and can be reused for deployment benchmarking. I then went further and "replayed" those traces to combine the advantages of simulation and experimentation with Trace-Based Simulation. I explain this work in Chapter 6. My goal is for the community to adopt a standard trace format and build an extensive set of traces to be able to test and validate protocols in a wide range of scenarios.

### 2.4.2 Determinism in IoT

Determinism is the theory according to which, given a system state, if an event occurs it results in an expected system state. In computer networking, this translates into the ability of predicting the performance of a network. Key metrics are typically the lifetime of a network (i.e. the time before one node runs out of battery), its reliability (i.e. how many messages were lost), and its latency (i.e. how much time messages need to go from source to destination). When a network is said to be deterministic, it means that it can provide guarantees. That is, network operator can commit to contracts named "service-level agreements" (SLAs) to ensure a client that the system will provide the desired quality of service (QoS)**gaillard14sla**, **gaillard14service**

Network Determinism has been around for years. Formed in 2012, the Time-Sensitive Networking (TSN) IEEE Task Group aims at providing deterministic services (not only time-related) through IEEE802 networks. Then the Deterministic Networking (DetNet) IETF working group started in 2014 and aims at providing networking layer determinism **ietf-detnet-architecture-07** They work with the TSN task group to define a common architecture for both layers. The DetNet working group also works with other IETF working groups such as 6TiSCH.

We saw that TSCH can provide high reliability together with low energy consumption. As TSCH uses a time slotted structure, nodes know when to sleep, transmit or receive, thus, most of the network events are known in advance. The uncertainty comes from the radio environment variations or the potential traffic changes. We will now see how performances guarantees can be obtain when using TSCH-based networks in term of reliability, latency and energy consumption.

**Reliability**

End-to-end (E2E) reliability is evaluated by the number of messages that reach their destination over the number of messages sent. For instance, a 90% E2E reliability means that, out of the 100 messages the were generated in the network, only 90 reached their destination. Message loss usually occurs for three reasons:

- Queue full. When a node receives a message and the node's message queue is full, the node cannot handle more messages and thus drops the new message.
- Maximum number of retransmissions. When a node tries to transmit a message, if the transmission fails, the node increases a retransmission counter. If a maximum number of retransmissions is set and the counter reaches that limit, the node discards the message.
- Disconnection. If a node has messages in its queue and it disconnects (e.g. due to high desynchronization) or runs out of battery, the messages are dropped.

A high E2E reliability (>99.99%) can be obtained by reserving enough resources (no queue full), allowing an infinite number of retransmissions (no retransmission limit reached), and ensuring low-power consumption.

In practice, reserving enough resources a complex task, especially when the number of retransmissions is unlimited. The number of resources (cells) required depends on the number of messages a node needs to transmit (locally generated messages and forwarded messages) but also on the expected number of retransmissions. Let's consider the scenario depicted in Fig. 2.9 where node `A` needs to transmit a message to node `B` at a rate of one message per slotframe, and there is one cell allocated from `A` to `B`. If the first transmission fails, node `A` can save the message in its queue and wait for the next available slot. As only one slot is available per slotframe, node `A` waits for the next slotframe and now has two messages to transmit and only one cell available. If retransmissions occur multiple times, the node's transmission queue might fill up and newly generated messages will be dropped. One way of solving

this problem is to allocate more cells, to anticipate retransmissions that may occur.

The question is: how to estimate the number of retransmissions? This boils down to: how to estimate the quality of a link ? Nouha Baccour *et al.* **baccour12radio** provide an extensive survey of the existing techniques to estimate the quality of a link. Rather than reproducing this survey, we list here the different paradigms and give some metric examples. First, we can discern between *passive* and *active* estimation.

In passive estimation, a node simply listens to the radio medium and estimates the quality of its different links from what it hears. This is the case when using the Received Signal Strength Indicator (RSSI) that denotes the amount of power a node receives when another node transmits. Using passive measurement is handy as it does not require a node to transmit any information. It does, however, rely on other nodes transmitting frequently in order to have up-to-date link information.

In active estimation, a node transmits frames to estimate the probability of success or failure of a transmission. The Packet Reception Rate (PRR) is the ratio of the number of packets successfully received over the number of packets sent. A similar metric is the Estimated Transmission Counter (ETX) **couto03high** that represents the expected number of transmissions needed to successfully transmit a frame. It takes into account the probability of delivery and the probability of "reverse delivery" (i.e the ACK probability of success). Such metrics usually provide a more precise understanding of the expected number of transmissions needed on average but do not take into account the maximum number of consecutive successes or failures.

Kannan Srinivasan *et al.* **srinivasan08beta** show that links are bursty, that is, they fluctuate between low and high delivery ratios and that existing metrics such as PRR does not denote those fluctuations. They introduce the $\beta$-factor, a metric to quantify the burstiness of a link. Knowing the number of estimated **consecutive failures** allows to estimate the number of cells needed in the worst-case conditions.

Pottner *et al.* **pottner14constructing** use $B_{max}$ that indicates the maximum number of retransmissions required to successfully transmit a frame. $B_{max}$ is obtained by exchanging data in the network, thus $B_{max}$ is not representative during the first exchanges of data but gets more and more accurate as the time goes. The authors use this metric to create a schedule for time-critical applications by allocating enough cells for the schedule to

Allowing an infinite number of retransmissions can result in very high reliability but has an impact on energy consumption as more cells are needed to enable retransmissions.

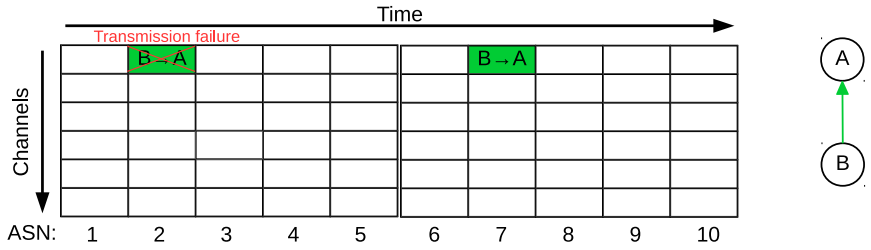



Figure 2.9: Time Slotted Channel Hopping schedule. A retransmission example. As only one cell is allocated per slotframe, nodes need to wait for 1 slotframe to retransmit.

### Energy Consumption and Latency

If the data traffic is periodic and we know the network topology, average link quality, and payload size, estimating the lifetime and latency of the network is straightforward **watteyne15industrial** The energy consumption is directly linked to the number of cells used as the main source of consumption is the radio. The latency depends on how the schedule is built. As the schedule is typically done as a function of the amount of data to carry (bandwidth needed) and the link quality (how many retransmissions needed), if those parameters are fixed and known in advance, we can estimate latency (as it is done in Ines Khoufi *et al.* **khoufi17scheduling**) and energy consumption (using models such as the one proposed by Xavier Vilajosana*et al.* **vilajosana14realistic**). Those conditions may not be met.

The first condition is to have periodic traffic. This depends entirely on the type of application. We can distinguish two types of application: **Constant Bit Rate**, where each node periodically generates data, and **Event-based**, where events are generated sporadically. In the latter case, for instance if alerts information is needed (e.g. button pressed, value threshold reached) and no resource (i.e. cells) is reserved in the schedule, the Scheduling Function needs to dynamically adapt the schedule (by triggering 6P events) to increase the number of allocated cells. This takes time, and the alert might not be relevant anymore when reaching the destination. The only way to reduce the resource reservation delay is to consider the worst case scenario and over-allocate resources, and thus increase the energy consumption of the devices.

The second condition is to have fixed link quality. Using channel-hopping averages the per-channel link qualities resulting in very stable links **watteyne09reliability** As an example, we observe links over days in our deployments **brun16intuitive** and show that routing with at most 5 parent changes per day can be achieved with highly reliable WSN.

**Toward Time-Critical Applications**

Wire-like E2E reliability is usually the first requirement that is demanded in industrial applications. Today, commercial products provide such reliability guarantees, we thus consider it as a solved issue. Wireless sensor network are now being studied in applications that require time-critical data collection.

Pöttner *et al.* study time-critical applications over TSCH networks in an oil refinery **pottner14constructing** They propose a metric called $B_{max}$ to quantify the maximum number of consecutive transmission failures to expect on a link, and based on that metric, they estimate the end-to-end (E2E) upstream latency of the packets. The drawback of this technique is that it requires several hours to have an accurate $B_{max}$ for each link.

Chang *et al.* propose a low-latency scheduling function (LLSF) for fast delivery in WSN**chang16llsf** They show that it is possible to achieve an average of E2E upstream latency of 320 ms over a 5-hop network.

Schindler *et al.* implement what they claim to be the first closed-loop wireless feedback control network using completely standards-compliant IEEE802.15.4 TSCH technology **schindler17implementation** They study the trade-off between duty cycle and latency in various combination over a 4-hops network.

Finally Yang *et al.* build an event-detection system using the 6TiSCH stack and study the latency distribution **yang18analysis** between two mote using different duty cycle. Their results are promising and show that per-link low-latency (tenth of ms) is achievable using 6TiSCH.

There is still a lack of understanding about how TSCH behaves in different environments and what are the trade-offs between, reliability, latency and network lifetime. In Chapter 6, we estimate the trade-offs between latency and energy consumption taking those conditions into account.

## 2.5 Summary

The IoT world is vast and its applications are flourishing. In this thesis, we focus on Industrial IoT, that is applications that should provide very high reliability event in harsh environments. Battery powered wireless technologies are more and more used in IIoT as they are easy to deploy and cost effective. To allow devices constrained in energy and computation power to operate in such harsh environments, the right technologies need to be selected. A wide range of standards exists but until recently no fully standardized networking stack existed to answer the IIoT requirements. The 6TiSCH stack was created to bridge that gap and will soon be fully approved. I believe we are at a cornerstone of Industrial IoT and that in the next following years we will see 6TiSCH adopted by a large number of network operators. Yet, some challenges remain. To be able to provide performance guarantees, the 6TiSCH

technology needs to be studied in depth over a wide range of applications scenarios and environments. In Chapter 6, I explain how I study the limits and trade-offs of TSCH, the mechanism at the heart of the 6TiSCH stack.

# Chapter 3

# Methodology and Assumptions

**In this chapter, we describe the methodology we follow to conduct our work. We start by deploying real-world IoT solutions and gathered what we believe to be one of the largest real-world IoT datasets available. Based on this ground work we propose improvements and recommendations for the IIoT.**

Our research methodology is not the usual one as it does not follow the typical order: proposition-measurements-validation. We believe that to be able to fully understand a system and its limits, we need to "get our hands dirty". We thus started by the measurements, gathering data and insights from real-world and testbed deployments. This allows us to ground or work in realistic assumptions, and produce solid results based on extensive datasets. From this ground work, we identify the limits and trade-offs of the networks we study, and propose enhancements. We then test and validate our proposals through simulation and experimentation.

## 3.1 Real-World Deployments

What is the state-of-the-art in IIoT and what are the *real* challenges the IIoT is faced with?

What do we mean by real-world networks? We consider a real-world network as a network that serves a real application, that is, an application whose purpose is not (or not only) to collect network statistics or test a system. Note that the network environment is not mentioned in that definition. There is neither a "right" nor a "wrong" network environment, as low-power wireless applications are very diverse.

While simulation and emulation can help proving theoretical protocol designs, they quickly face limits. There is a trade-off between realism and complexity: fully simulating the radio environment would lead to overly complex computation, but using simple models often does not represent reality. A common model type is the *Path-Loss* model that describes the signal attenuation between a transmitter and a receiver as a function of the propagation distance, on top of which other parameters can be added such as carrier frequency, antenna positions or terrain profile. Models such as the *Path-Loss* model rarely take into account multi-path fading or the variation of the radio environment over time.

Deploying hardware is more costly and creates systems that represent a single environment. A smart building deployment will always be an indoor deployment and the location of the devices will not change to adapt to a given application setup and then to another. Results obtained using testbeds are however more trustworthy than simulation as they are proven to work on real physical hardware (e.g. processing and memory delays and limits) and environments (radio propagation and interferences). Many testbeds are now being deployed and used to validate protocols. Although testbeds are more realistic, we show in Chapter 5 that they might not show the worst case characteristics of a real-world network and that results obtained from testbeds might not be sufficient to fully validate a protocol.

Deploying a network that serves a purpose other than testing protocols (i.e. a real-world network) helps ensuring a high level of realism, from the position of the devices, to the current consumed, the radio environment and the data traffic. We thus decided to start with real-world deployment to better understand the WSN and their challenges. I actively participated in the deployment of four real-world WSN deployments:

- **PEACH**: A frost prediction system for peach orchards in Mendoza (Argentina). We deployed 23 devices on a $11km^2$ area to measure temperature and humidity at different locations in the orchard.
- **SnowHow**: A snow-pack monitoring system in the Sierra Nevada (California). We deployed 27 devices in a $100km^2$ area to measure snow level, solar radiation and other environmental data in the mountain.
- **EvaLab**: A Smart Building monitoring system in Paris (France). We deployed 22 devices in a $800m^2$ area to measure temperature in an office building.
- **SmartMarina:** A metering and management system for marinas in Agde (France). We deployed 19 devices in a $5km^2$ area to measure the presence and electricity consumption of the boats in the marina.

As no end-to-end solution existed that suited our needs to format, transfer and store our data, we created SolSystem. We took existing tools and assem-

bled them, together with custom Python scripts, to create a system that is able to:

- **format data**: as some deployments are connected to the Internet using a satellite connection, we needed to highly compress the data produced by the network. We designed the concepts of "SOL Objects", a way of representing data in a compressed way, to which Object Security can be applied **vucinic15oscar**
- **transfer**: as we wanted to centralize the data of each deployment for easy treatment, we created a system to transfer the data from the gateway on the deployment site to a remote server.
- **store**: to make sure we do not loose any information, data is compressed and stored in both the gateway and the remote server.
- **display**: we use existing tools to visualize to visualize the data in real-time. This allowed us to have a clear view of what is happening in the field, in real-time.

These deployments gave me two things: real-world insights, and a large amount of data. We verified intuitive assumptions we had about real-world deployments such as network density, the relation between RSSI and distance, and overall network performances (reliability and lifetime). We went further and obtained not-so-intuitive results about link asymmetry and network churn.

We describe the deployments and their results in Chapter 4.

## 3.2 Characterizing Networks

How to characterize the radio environment a network is deployed in, and what are the key performance indicators to look at?

After learning about real-world networks, I wanted to get more insights from testbed deployments. As I started to understand the challenges of WSNs, the need for denser connectivity data increased. One key advantage of using testbeds is that it allows out-of-band communication. That is, devices are usually powered by cables and accessible using wired communication (e.g. serial communication). This allows users to program the devices by loading firmware, trigger commands to interact with the running system, or debug at run time by inserting breakpoints or observing logs. Out-of-band communication enables the collection of dense network statistics.

In real-world networks, we collect sensor data and network statistics. Network statistics are useful for the network to operate well, but should not reduce the resources for the sensor data to be carried over the network. Network

statistics are therefore usually sent at a low data rate. In our deployments, network statistics are generated by each device every 15 min, and contain averages of statistics over the last 15 min. To better understand the behavior of TSCH, I needed a denser statistics, i.e. more data.

Both testbed and real-world networks suffer from the same limitation: they only represent one scenario. In order to be properly validated, a system needs to be tested in all situations it is to be used in, or at least in the worst case scenarios. Testbeds are often considered realistic, but rarely represent a worst-case scenario. After observing testbeds that are part of the IoT-LAB, we found that they present only low external interferences from other technology (e.g. WiFi) and rarely show changes in radio connectivity. A protocol should not be validated by using only results obtained in such radio conditions unless the protocol is designed only for such conditions. This emphasizes the need for new ways of validating wireless protocols and increase testbed diversity.

To get those values I use the IoT-LAB, 6 large testbeds deployed throughout France, and designed for network design and testing. To collect connectivity data that is dense in time, space, and frequency, I modified and used a system called Mercator. I describe each experiment and its results in Chapter 5. Using Mercator, I collected radio connectivity traces that allow me to know the quality of the radio links between two devices in every available radio channels, and this, several times per minute.

In order to compare the datasets obtained from SolSystem and Mercator, I designed K7, a simple file format (extended CSV) that allows to present both types of data in a uniform way.

By extracting information from both testbed and real-world results, we were able to identify key parameters to look at when characterizing a network environment. We identified 3 phenomena that are the most common in real-world deployments, and which have a deep impact on connectivity: external interference, multi-path fading, dynamics in the environment. We consider those as the worst-case radio environments, and claim that a WSN protocol should be tested under such conditions (at least) in order to be validated.

Analyzing those radio connectivity traces and network statistics further improved my understanding of how the radio environment behaves.

## 3.3 TSCH Limits and Trade-offs

> Given a required end-to-end reliability, what is the minimum possible latency when using TSCH? How many years does the network loose if I want to reduce the end-to-end upstream latency by $X$ seconds ?

The Industrial IoT technologies are now adopted by many companies over the world and are now a proven solution. Yet challenges remain and some

of the limits of such technologies are still not fully understood. The use of TSCH-based networks to support time-critical applications is still at its early stages. More specifically, the trade-offs between latency and energy consumption (lifetime) are still not well understood. In this work, we try to fill that gap and study the limits of TSCH-based networks in term of:

1. **end-to-end upstream latency**. The time for a message generated by a sensor node to be delivered to the gateway.
2. **energy consumption**. How much energy is required to achieve a given latency limit.

To do so, we simulate networks on top of the connectivity traces obtained from both real-world and testbeds experiments. The connectivity traces replaces the (typically used) radio propagation models and provide results based on real-world radio environment. We recommend trace-based simulation as a tool for protocol performance evaluation.

## 3.4 Summary

In this chapter, we detail our methodology and assumptions. We explain how we go from exploring real-world deployments, to studying the performances limits and trade-offs of TSCH-based networks through trace-based simulation. We gather network statistics from real-world deployments from which we extract a set of conditions that we believe should be present to have representative results. We then study the limits and trade-offs of TSCH-based network in term of reliability, latency and energy consumption. By following a non-traditional approach, we believe we have identified realistic challenges and assumptions to propose results that are both representative of a real deployment, and immediately useful.

# Chapter 4

# Real-World Deployments

**In this chapter, we present the real-world IoT networks we deployed, and their corresponding applications. We explain how we collect network statistics and list the lessons we learned when deploying and operating networks in real-world conditions.**

"*C'est en forgeant que l'on devient forgeron*[1]". This rather old-fashion sentence, or at least the idea it carries, brought me to start my research by deploying wireless sensor networks. The first deployment happened to be near Mendoza, in Western Argentina. In April 2016, together with an international team from Chile, Argentina and France, I went to a city named Junín to deploy 23 nodes in a peach orchard to build a frost prediction IoT system. This first real-world deployment taught me a lot, from sensor integration and devices positioning to system architecture and data gathering. I then continued with other deployments in the Sierra Nevada in Eastern California, and in Agde, Southern France. In this Chapter, I describe each of these deployments and explain the lessons I learned.

We needed a common technology, both robust and already available on the market in order to understand development status of Industrial IoT. We selected SmartMesh IP. We describe how it works, and explain why we chose it in Section 4.1. We needed a system to store and analyze data produced by SmartMesh IP. No such system was available on the market, so we built our own, named SolSystem. We describe it in Section 4.2. The remainder of the chapter describes each real-world application, and the corresponding network we deployed. For each application, we present its context, describe our deployment, give an overview of the network performances, and list the

[1] literally: "its by forging that one become a blacksmith", and usually translated into "practice makes perfect".

lessons we learned. We first present two outdoor applications, Smart Agriculture in Section 4.3, and Environmental Monitoring in Section 4.4. These two applications have similar radio environment, but different scales (covered areas). We then present the EvaLab application in Section 4.5. In this indoor "Smart Building" application, radio communications are subject to external interferences and multi-path fading. We finish by presenting SmartMarina (Section 4.6), a marina monitoring solution, an outdoor application with large moving objects and external interferences. We compare the network statistics of those different networks in Chapter 5.

## 4.1 SmartMesh IP

### 4.1.1 An IIoT World Leader

For the first phase of this work we need a *product* that is thoroughly tested, proven and commercially available off-the-shelf. SmartMesh IP, a product line from the Dust Networks group at Analog Devices[2] fits those requirements. Dust Networks is initially a spin-off company from the University of California at Berkeley, now the world leader in supplying low power wireless mesh networks for demanding applications. Over 76,000 SmartMesh networks are deployed today in applications including industrial process monitoring, city-wide parking management, building automation and remote environmental sensing.

The protocol stack implemented in SmartMesh IP devices is standards-based, and includes standards such as IEEE802.15.4 Time Slotted Channel Hopping (TSCH) and IETF 6LoWPAN **watteyne13technical** A SmartMesh IP network is composed of one manager and up to 100 motes. The manager serves as the gateway of the SmartMesh IP network, and is typically connected to a computer, itself connected to the Internet. A mote is a standalone device, typically battery powered, itself connected to sensors and actuators. The SmartMesh IP network offers bi-directional connectivity: the motes can send sensor measurements to the gateway (and from there to some server on the Internet), the server can send actions/configurations to the motes.

Security is built into the SmartMesh IP protocol stack around an AES-128 cipher, and cannot be switched off. A set of keys ensures confidentiality, integrity and authentication of the data. The SmartMesh IP network is operated in such a way that it offers wire-like reliability, with over 99.999% end-to-end reliability.

Each node in a SmartMesh IP network can periodically send beacons to announce the presence of the network. When a mote wants to join a network, it listens for those beacons. Once it has heard a number of those, it

[2] `https://www.analog.com/`

starts a security handshake with the network. During that handshake, the SmartMesh IP manager sends a `mote_create` event notification over its serial port. It contains, among other information, the association between the newly-joined device's 8-byte MAC address and its 2-byte `moteId`.

In SmartMesh IP terminology, a "path" is the link-layer resource that allows two neighbor nodes to communicate[3]. Each time a mote starts communicating with a new neighbor (e.g. its routing parent), a `path_create` event is produced. Similarly, each time a mote *stops* communicating with a neighbor (e.g. it changes routing parent), a `path_delete` event is produced. We log both messages.

The motes in a SmartMesh IP network automatically and periodically generate "Health Reports" (HR), a series of counters and statistics to assess the overall health of the network. Three types of statistics are generated:

- `HRDevice`. Internal counters of the mote (e.g. packet counters, energy consumed)
- `HRNeighbors`. The list of neighbors the mote is communicating with
- `HRDiscovered`. The list of neighbors the mote can hear but is not communicating with.
- `HRExtended`. The number of transmission attempts and link-layer retransmissions *for each frequency*.

A mote generates a complete set of health reports every 15 min.

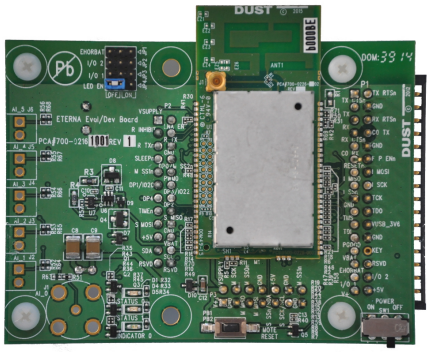
A SmartMesh IP network is fully synchronized, and time is split up into timeslots. A schedule orchestrates all communication in a SmartMesh IP network and indicates what to do in each timeslot: transmit, listen or sleep. How the schedule is built and maintained allows a clear trade-off between the amount of data generated by the sensors, the communication latency, and the power consumption of the motes.

The "Dust Networks SmartMesh Power and Performance Estimator" is a tool provided by Dust Networks[4] that allows one to estimate the power consumption and latency of a SmartMesh IP network, given the topology and amount of data generated. A typical SmartMesh IP network offers over 99.999% reliability and over a decade of battery lifetime when motes are powered by a pair of AA batteries **watteyne15industrial**

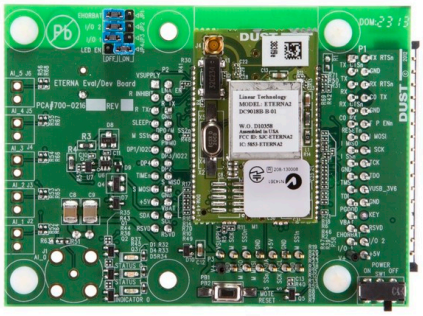
[3] In more classical networking terminology, this is often referred to as a "link". We use the terms "path" and "link" interchangeably in this work.

[4] `http://www.analog.com/en/technical-articles/smartmesh-power-and-performance-estimator.html`

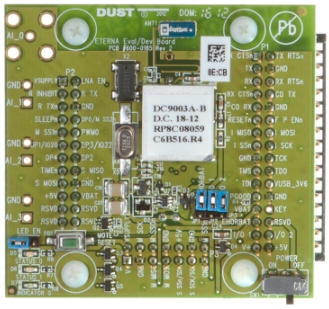
(a) DC2274: Low-power wireless manager *(1 deployed)*



(b) Long-range prototype board *(3 deployed)*



(c) DC9018: External antenna mote *(2 deployed)*



(d) DC9003: chip antenna mote *(16 deployed)*

Figure 4.1: The boards deployed.

### 4.1.2 Low-power Wireless Motes

Fig. 4.1 shows the three types of motes that we use in our deployments. All boards are manufactured by Analog Devices. DC9003 (Fig. 4.1d), DC9018 (Fig. 4.1c) and long-range prototype boards (Fig. 4.1b) are deployed.

The DC9003 (Fig. 4.1d) is the development board for SmartMesh IP networks. This is the board used to experiment with and prototype SmartMesh IP technology. All the pins of the micro-controller are exposed, allowing a developer to interface sensors and actuators over digital and analog interfaces. The DC9003 features the LTC5800 chip (a SoC with an ARM Cortex-M3 and an IEEE802.15.4 radio) connected to a chip antenna. The DC9003 is an off-the-shelf commercially available board.

The DC9018 (Fig. 4.1c) is functionally equivalent to the DC9003, but

with a 2 dBi external antenna, rather than a chip antenna. It is expected that the DC9018 has a better range than the DC9003. The DC9018 is an off-the-shelf commercially available board.

The long-range board depicted in Fig. 4.1b is a prototype of a new product Analog Devices allows us to test. It is a fully SmartMesh IP-compliant product (i.e. it interoperates with DC9003 and DC9018 boards in the same network), with a line-of-sight range of 1500 m. This board features a 2 dBi external antenna.

The boards deployment in Phase 1 of the PEACH project are developer/prototyping boards. We expect to transition to the NeoMote in Phase 2, a ruggedized version of the motes, manufactured by Metronome Systems[5].

All the boards deployed during Phase 1 run the default SmartMesh IP firmware. Each board automatically joins the network and sends a temperature reading every 30 s to the manager, using its built-in temperature sensor. In Phase 2, we will reprogram the motes with custom firmware, using the "DustCloud" development environment provided by Analog Devices[6].

### 4.1.3 Low-power Wireless Manager

Fig. 4.1a shows the SmartMesh IP manager used in the PEACH project. It is composed of an LTP5902-IPR SmartMesh IP manager chip, a 2 dBi external antenna and USB connectivity, in the form-factor of a USB pen drive. USB connectivity is used to connect the DC2274 to a computer. The computer can run a program to configure the DC2274, send commands and configurations to the motes, and retrieve sensor measurements. The DC2274 is a standalone device, i.e. the computer does not play any active role in operating/managing the network, it just serves as an interface between the SmartMesh IP network and the Internet. The DC2274 is an off-the-shelf commercially available board.

## 4.2 SolSystem

SmartMesh IP provides the networking solution and makes sure data is carried to the gateway. Then data needs to be forwarded to other entities in order to be stored and analyzed. Analog Devices provides a set of applications and libraries to start manipulating the network and its data but does not provide an "End-to-End Solution" that allows to carry, store, analyze and display this data. Other solutions exist such as ThingWorx, Xively or ThingSpeak but do not provide the flexibility we wanted. As we will see in Section 4.4, some deployments need to highly compress the data generate as the only available connection to the Internet is done via satellite, with a very limited data plan.

[5] `http://metronomesystems.com/`

[6] `http://www.dustcloud.org/`

We hence needed a solution that allowed us to use our own optimized compression format. Another limitation comes from the fact we have two types of data, sensor data and network statistics, and solutions such as the ones presented above are only focusing on sensor data. Current solutions are not flexible enough or do not match the types of data we are manipulating in our work. We hence developed our own solution named SolSystem.

SolSystem is composed of two entities: the `solmanager` and the `solserver`. The `solmanager` lives on-site, and directly communicates with the SmartMesh IP manager. It collects, stores and forwards the data produced by the network to the `solserver`, that lives in a remote computer.

### 4.2.1 `solmanager`

The `solmanager` application runs on an embedded computer (e.g. a Raspberry Pi computer) and connects to the serial port of the SmartMesh IP manager. It subscribes to all the notifications from the SmartMesh IP manager: data, health reports, network events. It then parses that information and formats it as a series of SOL objects, which it forwards to the `solserver` application.

In the PEACH deployment, a Raspberry Pi is connected to an Ethernet-to-WiFi bridge equipped with a long-range directional antenna pointing to a building with WiFi Internet access 200 m away. In practice, this means the Raspberry Pi is directly connected to the Internet. Through that connection, the Raspberry Pi forwards all the generated SOL objects, in binary format, to the `solserver` application over HTTPS. Public/private Transport Layer Security (TLS) keys and an HTTP token provide confidentiality, integrity, authentication and authorization of the `solmanager` $\leftrightarrow$ `solserver` communication.

### 4.2.2 `solserver`

The `solserver` application runs on a server in the Inria-Paris research center. The application listens for incoming SOL objects. When those objects are received from a properly authorized `solmanager` application, the `solserver` converts them from their binary representation to their JSON representation, and writes them into a database[7].

A dashboard allows a user to visualize key information. Fig. 4.2 shows an example dashboard with 7 days worth of data, including the temperature (the day/night temperature swing is clearly visible, the last day being cold), the number of discovered neighbors, and the charge consumed by the devices.

[7] We use an InfluxDB (`https://influxdata.com/`) database.


Figure 4.2: Example dashboard showing the last 7 days of data.

### 4.2.3 SOL

The SmartMesh IP network continuously generates information: the data produced by the motes, the Health Reports produced by the motes, and network events (e.g. a mote joined the network) produced by the manager. The data produced by the motes contains the bytes generated by the application running on the mote; this data is formatted differently depending on the application.

We created the "Sensors Object Library" (SOL) to wrap that information as generic SOL objects. A SOL object is always composed of the same fields:

- the **identifier** of the device generating the information,
- the **timestamp** of when this information was generated,
- the **type** of information,
- the **value** of the information.

A SOL object can be seen as a generalization of the Type-Length-Value (TLV) format.

A SOL Registry contains the list of SOL types currently used, and their associated format. This includes data generated by the motes, health reports, and network events. Each SOL object can be represented either in a compact binary form, or in a more easily parseable JSON format. The SOL registry

is publicly maintained and can be easily augmented with new types of sensor data[8].

We use SolSystem in each deployment we will present in the following sections. SolSystem is flexible enough to match each one of them and greatly helped us in the deployment process, the monitoring process and the analysis process. We will continue using it in our future deployments.

We will now go through each of the real-world deployment we conducted. For each deployment we present its application context, describe the deployment, give an overview of the network performances, and list the lessons we learned.

## 4.3 PEACH

A frost prediction solution for peach orchards in Mendoza (Argentina).

### 4.3.1 Context and objectives

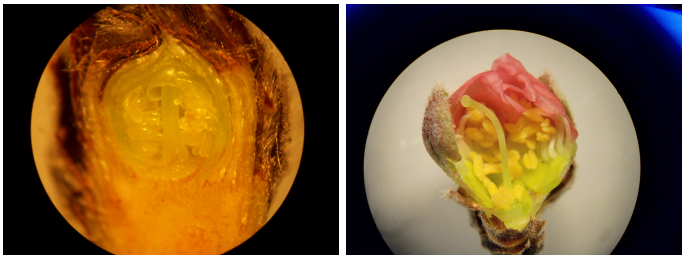
In 2013, 85% of the peach production in the Mendoza region (Argentina) was lost because of frost. Because less fruit was produced in the region, 600,000 less work days were needed to process the harvest between November 2013 and March 2014, a reduction in work force of 10,600 people. Across the Mendoza region, frost caused a loss of revenue of 950 million Argentine pesos, roughly 100 million USD (at that time) in the peach business alone. Handling a frost event is possible, but it is hard to predict when it is going to happen. The goal of the PEACH project is to predict frost events by analyzing measurements from sensors deployed around an orchard. This section provides an in-depth description of a complete solution we designed and deployed: the low-power wireless network and the back-end system. The low-power wireless network is composed entirely of commercial off-the-shelf devices. We develop a methodology for deploying the network and present the open-source tools to assist with the deployment and to monitor the network. We discuss how the technology used is the right one for precision agriculture applications.

Peach (*Prunus persica* (L.) Batsch.) is a deciduous fruit tree, growing mainly in temperate regions. It has a rest period between vegetative cycles, saving growing organs from extreme winter frost. To leave winter dormancy, buds need to accumulate chill followed by heat, resulting in Spring flowering **chaar12determinacion**

Peach flower buds contains floral primordia, consisting of sepals, petals, androecium and gynoecium (see Fig. 4.3).

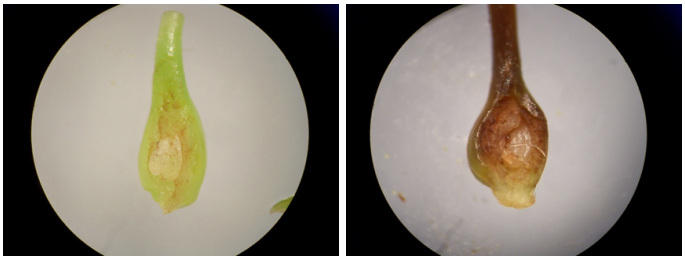
Spring frost events are one of the main limiting factors for the production of temperate fruit trees **chaar2015caracterizacion** To develop into a fruit,

[8] `https://github.com/realms-team/sol/blob/master/registry.md`



(a) Flower bud on vegetative rest (b) Flower bud prior to spring blooming

Figure 4.3: Peach flower buds at different stages of winter rest. The flower bud has a lower resistance to frost in (b).



(a) healthy (b) damaged

Figure 4.4: Gynoecium healthy and damaged by frosts. Damage is visualized by brown tissues (extracted from **chaar2015caracterizacion**).

a flower bud needs to bloom without frost damage, to later be successfully pollinated and fertilized, eventually causing fruit set. When the temperature drops below zero in Spring, the migration of water to inter-cellular spaces causes tension and breaks the cell membranes. This causes internal solutes to be lost and cells to die.

Fruit production mainly depends on the number of healthy flowers that resist subzero temperatures. Different peach cultivars have different levels of flower bud resistance to Spring frost, depending on the date of full flowering, the lethal temperature, and the flower density **chaar2015caracterizacion** see Fig. 4.4.

In Mendoza, Argentina, there are 14,216 ha of peaches, including 5,759 ha for fresh consumption **IDR-censo2010** and 8,457 ha for the canning industry **IDR-2014** During Spring 2013, a series of frost events (see Fig. 4.5) caused major damages in the peach production. 56% of the fruit production area was affected by frost. It is estimated that 85% of the peach production in the Mendoza region was lost. Because less fruit was produced, 600,000 less work days were needed to process the harvest between November 2013 and March 2014, resulting in a reduction in work force of 10,600

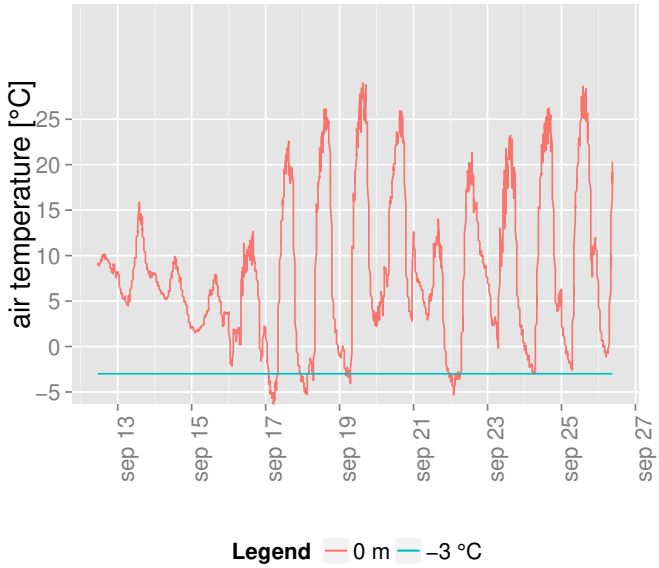



Figure 4.5: Air temperature at ground level measured in Junín, Mendoza in Spring 2013, showing several frost events.

people. Across the Mendoza region, frost caused a loss of revenue of 950 million Argentine pesos, roughly 100 million USD (at that time) in the peach business alone **DACC-2014**, **IDR-censo2010**).

A frost event happens when the temperature is so low that the crops cannot recover their tissue or internal structure from the effects of water freezing inside or outside the plant. For the peach production, a critical period is when the trees are in bloom and fruit sets (August/September in Mendoza), during which the temperature needs to be kept above −3 C. Even a few hours below that temperature causes flowers to fall, preventing fruit to grow.

Because of the huge economic impact, countermeasures exist, but are expensive. Today, virtually all industrial peach orchards are equipped with a meteorological station that monitors temperature and humidity. Farmers need to deal with false positives (the countermeasure is applied, but there is no frost event) and false negatives (the meteorological station does not pick up a frost event happening in some part of the orchard).

What is needed is a dense real-time monitoring solution deployed in the orchard, feeding a frost prediction model. Having a meteorological station does not provide the measurement density needed. Frost events are microclimatic: cold and hot air have a different density, wind blows irregularly between the trees, so different parts of the orchard are affected very differently by frost. What we build is a system with a large number of sensing points (air temperature, air relative humidity, soil moisture, soil temperature), at

different elevations/depths, throughout the orchard.

Low-power wireless mesh networking technology has evolved significantly over recent years. With this technology, a node is the size of a deck of cards, is self-contained and battery-powered. When switched on, nodes form a multi-hop low-power wireless network, automatically. Off-the-shelf commercial solutions are available today that offer $>99.999\%$ end-to-end data reliability and over a decade of battery lifetime.

The goal of the PEACH project is to increase the predictability of frost events in peach orchards by using dense monitoring provided by low-power wireless mesh networking technology.

### 4.3.2 Related applications

Active frost control is possible **snyder2005frost** One option is to position heaters across the orchard, but the fuel used is a pollutant and is expensive. Over-plant sprinklers can also be used (latent heat is produced when the water from the sprinklers freezes), but installation cost is high and a lot of water is needed. Another option is to mix the warmer air above the orchard with the cooler air at ground level, causing an increase in temperature around the flowers. This can be done by vertical wind towers, or even flying a helicopter above the orchard. Regardless of the method used, it is expensive. It is therefore essential to be able to accurately predict the frost event.

To predict a frost event, one needs to know the dew point. The problem is that the dew point depends on the relative humidity: at lower relative humidity, air temperature drops faster and to lower values. To predict the minimum temperatures – hence a frost event – one needs to observe the temperature at several locations across an orchard, at different elevations, since winds and foliage cause micro-climatic differences in the orchard. Using a single meteorological station in the orchard is not enough; the temperature and humidity it measures are not representative of the entire orchard.

To accurately characterize and predict frost events, what is needed are air temperature and relative humidity measurements at different locations across the orchard, at different elevations, as well as soil temperature and soil moisture at different depths. This data can then feed to a machine-learning algorithm to determine which features are important to predict frost events, and eventually build an early warning system.

### 4.3.3 Deployment

The goal of the PEACH project is to retrieve data from different types of sensors deployed densely across a peach orchard. Because heavy machinery is used throughout the field, using wires to interconnect the sensors is not an option. We instead use a low-power wireless mesh network. Our requirement is that the security and reliability of this wireless network must be equivalent

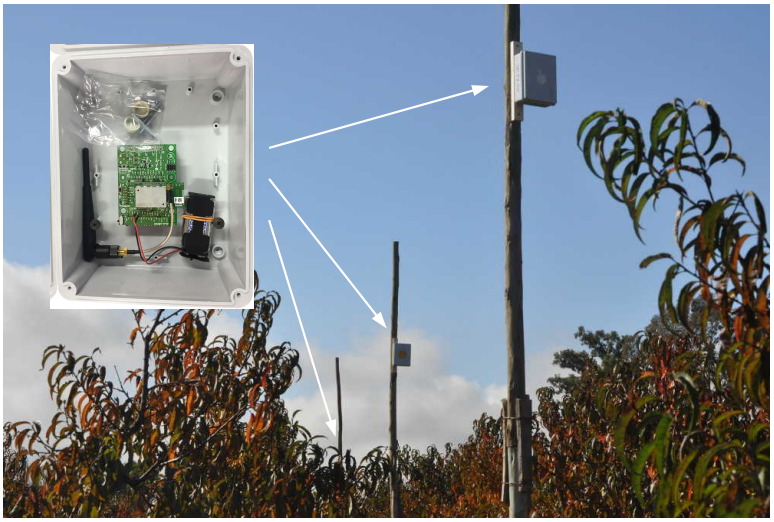


Figure 4.6: The wireless motes deployed in the peach orchard in Mendoza, Argentina.

to that of a wired network. Our target battery lifetime for the devices is "at least a year", the natural grow cycle of peaches.

This project is not a low-power wireless project, i.e. we need a technical solution that "just works". While very interesting research can be done with academic projects **mesas2015open**, **budi2015optimisation** (there are many in the research field of low-power wireless), we need a *product* that is thoroughly tested, proven and commercially available off-the-shelf. We have selected SmartMesh IP, which satisfies our requirements.

We deployed a SmartMesh IP network in a peach orchard of 204 trees, planted in a 50 m × 100 m area (shown in Fig. 4.7). The low-power wireless network is composed of 18 sensor motes[9] uniformly distributed between the peach trees, and 3 relay motes to connect the orchard to the gateway some 300 m away. Each mote is placed in a water-tight box that is fixed on a 4 m high pole (see Fig. 4.6).

### 4.3.4 Hardware Integration

The PEACH network is deployed in an orchard for 2 years, and is exposed to direct sunlight, freezing temperatures, dust and rain. All motes and manager are therefore protected by an enclosure with Internal Protection 65 (IP65) rating[10].

Fig. 4.8a shows the manager enclosure with the lid open. It contains the DC2274 board, a Raspberry Pi single-board computer running the

[9] 2 motes malfunctioned and are not present on the map.

[10] Totally dust tight and protection against low pressure water jets in all directions.

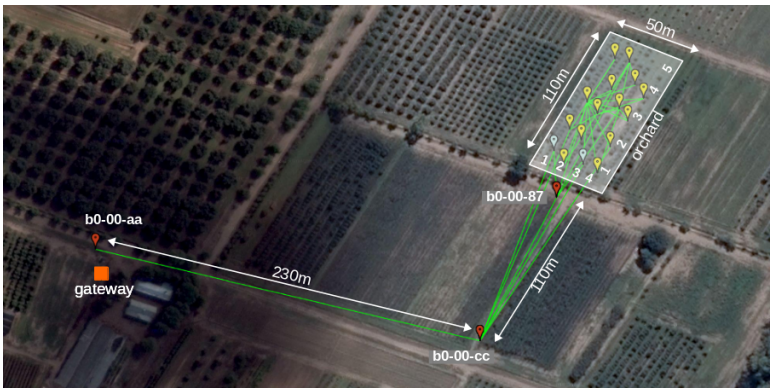



Figure 4.7: Areal view of the sensor network deployed in the orchard near Mendoza, Argentina.

(a) Manager (b) Mote

Figure 4.8: The devices encased in 20 cm × 15 cm IP65 enclosures.

| | |
|---|---|
| reliability | 100% (Arrived/Lost: 243089/0) |
| stability | 93% (Transmit/Fails: 1462435/96923) |
| latency | 800 ms |

Table 4.1: Key network performance indicators.

`solmanager` software (see Section 4.2.1), and the power adapter for that board. The DC2274 is powered through USB. A water-tight pass-through opening in the manager enclosure is used to pass power and Ethernet cables.

Fig. 4.8b shows a mote in its enclosure. The mote is powered by a pair of Energizer L-91 AA batteries with a charge of 2821 mAh[11]. The mote, antenna and battery are glued in place using silicone. No opening is present in the mote enclosures.

### 4.3.5 Performance of the Network

Table 4.1 summarizes the network performance after 5 days and 6 hours of operation. After approx. 24 hours of operation, a SmartMesh IP network reaches "steady state", so we expect the numbers presented in this section to be representative of the lifetime of the network.

The network **reliability** indicates the portion of the packets generated by the motes that reach their final destination, in our case the SmartMesh IP manager. A packet is said "lost" when it never reaches its destination. The motes in the PEACH deployment have generated 243,089 packets, all were successfully received by the SmartMesh IP manager, in part after multiple hops, yielding 100% reliability.

The wireless medium is unreliable in nature, and it is common that a link-layer frame sent by mote `A` is not received by mote `B`, forcing `A` to *re*-transmit. The network **stability** represents the average PDR over the link. The motes in the PEACH deployment have sent 1,462,435 data-link frames, 96,923 unsuccessfully (i.e. no link-layer acknowledgment was received), yielding a network stability of 93%. This number is very high, indicating that the nodes are deployed close enough in an environment with little external interference and multi-path fading.

A SmartMesh IP network is fully synchronized. A mote generating a data packet adds a timestamp to its data so that what is written in the database is the time the sensor is sampled (not the time it reaches the database). As a side effect, it is possible, at the SmartMesh IP manager, to calculate how long the packet has been traveling in the multi-hop wireless mesh network. This is called average network **latency**. In the PEACH deployment, it takes on average 800 ms for a sensor measurement to travel from the mote that

[11] The charge mentioned in the datasheet of the Energizer L-91 is 3135 mAh, but we assume a 10% derated charge to account for shelf-life.

| MAC | Uptime | Neig. | Cells | Hops | Latency | Recv'd | Lost | Relia. | Charge | Lifetime |
|---|---|---|---|---|---|---|---|---|---|---|
| 60-3C-D9* | 5-06:57:07 | 2 | 40 | *N.A.* | *N.A.* | *N.A.* | *N.A.* | *N.A.* | *N.A.* | *N.A.* |
| B0-00-AA | 5-06:56:06 | 3 | 20 | 1.0 | 170 ms | 16450 | 0 | 100% | 33263 mC | 4.42 years |
| B0-00-CC | 5-06:54:55 | 5 | 18 | 2.7 | 280 ms | 16657 | 0 | 100% | 28396 mC | 5.18 years |
| 58-32-36 | 3-09:29:11 | 10 | 21 | 3.0 | 420 ms | 10788 | 0 | 100% | 27546 mC | 3.43 years |
| B0-00-BE | 3-10:17:42 | 6 | 15 | 2.8 | 670 ms | 10560 | 0 | 100% | 15147 mC | 6.30 years |
| 60-06-0F | 3-09:01:05 | 4 | 11 | 3.5 | 1020 ms | 10386 | 0 | 100% | 15759 mC | 5.96 years |
| B0-00-87 | 3-10:06:30 | 6 | 37 | 1.2 | 120 ms | 10628 | 0 | 100% | 20038 mC | 4.75 years |
| 3F-F8-20 | 3-09:21:15 | 2 | 9 | 3.6 | 1180 ms | 10428 | 0 | 100% | 11907 mC | 7.92 years |
| 30-60-EF | 3-09:16:26 | 9 | 24 | 2.7 | 340 ms | 10551 | 0 | 100% | 22471 mC | 4.19 years |
| 60-03-82 | 3-09:11:22 | 4 | 11 | 3.5 | 740 ms | 10409 | 0 | 100% | 15977 mC | 5.89 years |
| 60-08-D5 | 3-09:04:03 | 2 | 9 | 3.5 | 810 ms | 10389 | 0 | 100% | 11173 mC | 8.41 years |
| 3F-FE-88 | 3-08:56:18 | 3 | 10 | 3.5 | 1210 ms | 10384 | 0 | 100% | 12924 mC | 7.26 years |
| 3F-FE-87 | 3-08:51:28 | 2 | 9 | 4.2 | 1440 ms | 10372 | 0 | 100% | 11900 mC | 7.88 years |
| 60-05-5F | 3-08:45:53 | 2 | 9 | 4.4 | 1860 ms | 10346 | 0 | 100% | 10867 mC | 8.61 years |
| 60-06-27 | 3-08:45:09 | 4 | 12 | 3.6 | 770 ms | 10368 | 0 | 100% | 14629 mC | 6.40 years |
| 60-05-69 | 3-08:40:17 | 2 | 9 | 3.6 | 1100 ms | 10334 | 0 | 100% | 10915 mC | 8.57 years |
| 60-01-F8 | 3-08:37:02 | 3 | 10 | 3.6 | 640 ms | 10322 | 0 | 100% | 11292 mC | 8.28 years |
| 60-02-4B | 3-08:31:59 | 2 | 9 | 4.3 | 1520 ms | 10326 | 0 | 100% | 13186 mC | 7.08 years |
| 60-02-1B | 3-08:28:39 | 6 | 14 | 3.5 | 650 ms | 10301 | 0 | 100% | 14700 mC | 6.35 years |
| 60-05-AB | 3-08:22:30 | 3 | 10 | 4.0 | 920 ms | 10298 | 0 | 100% | 12808 mC | 7.27 years |
| 60-06-EC | 3-08:21:17 | 2 | 9 | 4.4 | 1740 ms | 10289 | 0 | 100% | 10964 mC | 8.50 years |
| 38-0F-66 | 3-08:03:38 | 2 | 9 | 4.4 | 1430 ms | 10254 | 0 | 100% | 10781 mC | 8.61 years |
| 60-05-78 | 3-08:00:26 | 2 | 9 | 3.6 | 950 ms | 10247 | 0 | 100% | 10710 mC | 8.66 years |

* the manager

Table 4.2: Key mote performance indicators.

generated it to the SmartMesh IP manager, over the multi-hop low-power wireless mesh network. This latency is very small compared to the variation speed of the signal we are observing (temperature).

### 4.3.6 Performance of the Motes

The health reports generated by each mote contain a wealth of information. Table 4.2 contains a summary of the most important information for assessing the performance of the network.

Each line in Table 4.2 is indexed by the unique identifier of the mote, its "**MAC** address". This is a unique 8-byte number – called EUI64 – written into the mote at manufacturing time. The EUI64 of all motes starts with `00-17-0d-00-00`; in the interest of space, this 5-byte prefix does not appear in Table 4.2.

The **Uptime** is the time the mote has been operational, and is of format `day-hour:min:sec`. The 3 first motes were deployed 2 days before the others; it is hence normal to have a 2-day gap in their uptime.

Each node reports the number of neighbors it currently sees. This is shown in column "**Neig.**" in Table 4.2. The number of neighbors indicates the density of the deployment. Dust Networks recommends for each mote to ideally have 3 or more neighbors. This is not the case for 10 of the motes in the PEACH deployment. We expect that, when swapping the DC9003 boards (chip antenna) for boards with external antenna, the density will increase.

All communication in a SmartMesh IP network is orchestrated by a communication schedule. The column **Cells** in Table 4.2 indicates the number of cells in the schedule the mote is active in (transmitting or receiving). That number is directly proportional to the energy it consumes. We analyze the energy consumption more finely below.

Each packet sent through the low-power SmartMesh IP mesh network contains a hop count field[12]. This allows the SmartMesh IP manager to know how many hops this packet took to go from its source to the manager. The value printed in the **Hops** field is an average calculated over the packets received so far. As can be seen from Table 4.2, the closest (resp. furthest) mote is 1.0 (resp. 4.4) hops away from the manager. Dust Network recommends to keep that number below 8 for efficiency; a condition we satisfy.

The **Latency** column shows the average latency for a packet to go from that node to the manager. The average value of all motes is the network latency shown in Table 4.1. As can be seen, the latency increases with the number of hops.

The overall network reliability is shown in Table 4.1. Table 4.2 shows the same information, but broken up per mote. Column "**Recv'd**" shows the number of packets generated by that mote that have reached the manager. Column "**Lost**" shows the number of packets lost en route. No packets are lost, the **Reliability** for each node is 100%.

In their health reports, motes indicate the amount of **Charge** drawn so far from their battery, in mC. Knowing their uptime and the theoretical charge of their battery[13], we calculate the expected **Lifetime** of each mote. The expected lifetime varies between 3.43 years and 8.66 years, depending on the type of motes (a long-range mote consumes more than a DC9003) and its activity (a mote with more cells consumes more). These numbers are well above the targeted 1 year of lifetime.

### 4.3.7 After 3 Months

In 3 months of operation, we gathered over 4 million temperature values, and more than 350,000 network statistics **brun16intuitive**

Table 4.3 summarizes the number of events and HRs gathered during the 3 month period. In the remainder of this section, we detail the meaning of each of the statistics.

[12] Equivalent to the "hop limit" field in the IPv6 header.

[13] We assume 2821 mAh, per Section 4.3.4 of charge, or 101,556,000 mC.

| type | number |
|---|---|
| mote_create | 133 |
| path_create | 4,098 |
| path_delete | 3,653 |
| HRDevice | 132,758 |
| HRDiscovered | 87,737 |
| HRNeighbors | 140,897 |

Table 4.3: The number of statistics collected over the 3 month period.

### 4.3.8 Intuitive Results

Previous publications **watteyne16peach**, **watteyne10mitigating**, **watteyne09reliability**, **watteyne15industrial** underline the performance of TSCH networks in general, and SmartMesh IP in particular. Standardization work in the IETF 6TiSCH working group[14] around TSCH networks further illustrates the move of the industry towards this type of networking technology. So while we expect good performance from the network, we verify that this is indeed the case. We start by looking at two physical-layer metrics: RSSI vs Distance and PDR vs. RSSI. While these have no dependency on TSCH (the type of medium access), they allow us to verify the overall connectivity in the network. We then look at key performance indicators of SmartMesh IP networks: end-to-end reliability and network lifetime.

#### RSSI vs. Distance

The Friis transmission model **saunders07antennas** gives the relationship between the Received Signal Strength (RSSI)[15] in free space. While it does *not* apply directly to our Smart Agriculture outdoor deployment, we note in Fig. 4.9 that the individual RSSI values are located between the Friis model, and the Friis model offset by $-40$ dB. This corroborates the results from **zats10wireless**

#### Wireless Waterfall

Due to the inherent physical unreliability of the radio medium, it is impossible to know if a future transmission will be successful or not. The Packet Delivery Ratio (PDR) is the portion of successful link-layer transmissions over the total number of link-layer transmission attempts. A failed attempt

[14] `https://tools.ietf.org/wg/6tisch/charters`

[15] Strictly speaking, the RSSI is the Received Signal Strength *Indicator*, a value returned by radio chip. Because of its prevalence in low-power wireless literature, we use it RSS and RSSI interchangeably.

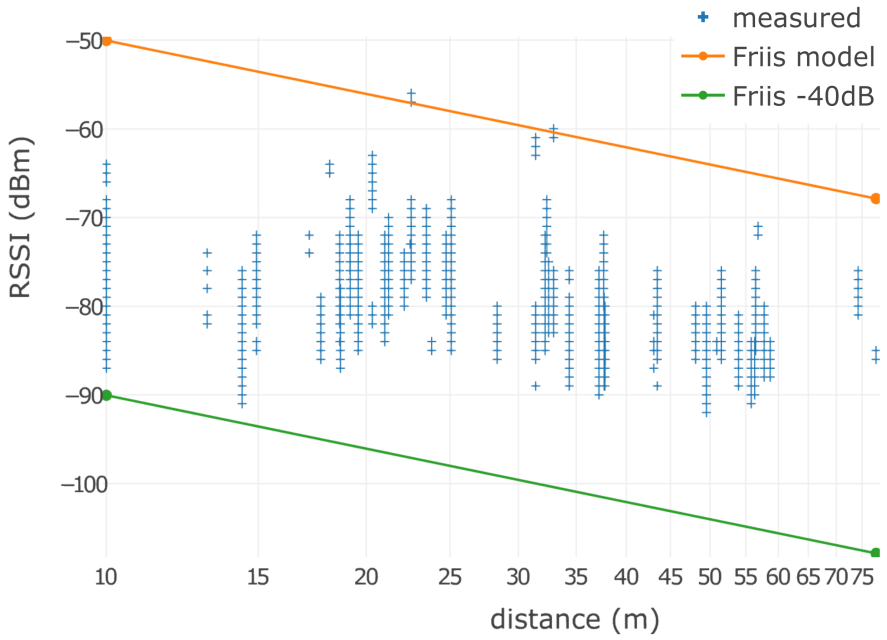



Figure 4.9: RSSI measurements are roughly located between the Friis model and the Friis model shifted by −40 dB.

means that the link-layer frame needs to be re-transmitted; it does *not* mean the packet is lost. Over a period of 3 months, 140,897 `HRNeighbors` messages are collected. These contain, for a given node, the number of link-layer transmission attempts and successes to each of its neighbors. We remove the portion of neighbors with no transmission and keep only the DC9003 motes, resulting in a total of 88,284 messages (approx. 37% from the total number of `HRNeighbors`).

Fig. 4.10 plots the PDR and the RSSI of these 125,103 messages. For readability, we also plot the average/deviation of the data for a given RSSI value. Because of its shape, this is known as the "waterfall plot".

Overall, above −85 dBm, the PDR of the link is very good (>95%). Below that value, the PDR rapidly degrades, indicating that, on these links, frequent retransmissions happen. The device manufacturer documentation **smip_app_note** indicates that a path is considered as "bad" when:

- RSSI>−80 dBm and PDR<50%
- RSSI>−70 dBm and PDR<70%

This is not the case here.

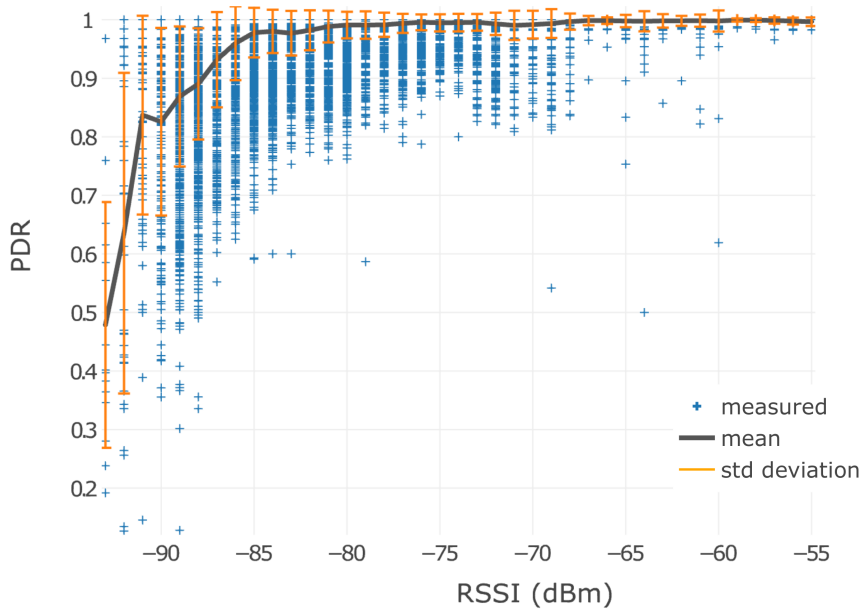



Figure 4.10: The PDR/RSSI "waterfall" plot.

| | |
|---|---|
| reliability | 100% (Arrived/Lost: 693844/0) |
| average PDR | 95% (Transmit/Fails: 4405569/258778) |
| latency | 700 msec |

Table 4.4: The overall network performance in the 15-25 July 2016 period.

A *waterfall plot* either shifted right with very few paths below −70 dBm, or with a non-constantly decreasing curve would be an example of interference-prone environment. This is not the case in Fig. 4.10, meaning that the SmartMesh IP network is not experiencing high levels of interferences from co-located wireless devices.

**End-to-End Reliability**

We expect the SmartMesh IP network to offer wire-like reliability. Table 4.4 confirms that this is the case. It presents statistics gathered over the 15-25 July 2016 period.

It shows that, as none of the 693,844 packets generated in the network was lost, the end-to-end reliability is 100%. The average PDR over all the links is very high (95%), indicating that the nodes are deployed close enough to one another. Finally, the average latency over all nodes

| MAC address | charge consumed | lifetime |
|---|---|---|
| 30-60-ef | 227,847 C (2.2% battery) | 10.8 years |
| 38-0f-66 | 252,356 C (2.5% battery) | 9.8 years |
| 3f-f8-20 | 291,312 C (2.9% battery) | 8.4 years |
| 3f-fe-87 | 392,606 C (3.9% battery) | 6.3 years |
| 3f-fe-88 | 458,459 C (4.5% battery) | 5.3 years |
| 58-32-36 | 327,634 C (3.2% battery) | 7.5 years |
| 60-01-f8 | 252,454 C (2.5% battery) | 9.8 years |
| 60-02-1b | 222,253 C (2.2% battery) | 10.1 years |
| 60-02-4b | 146,068 C (1.4% battery) | 16.8 years |
| 60-03-82 | 494,841 C (4.9% battery) | 5.0 years |
| 60-05-5f | 274,502 C (2.7% battery) | 9.0 years |
| 60-05-69 | 437,136 C (4.3% battery) | 5.7 years |
| 60-05-78 | 304,145 C (3.0% battery) | 8.1 years |
| 60-05-ab | 284,764 C (2.8% battery) | 8.7 years |
| 60-06-27 | 321,879 C (3.2% battery) | 7.7 years |
| 60-08-d5 | 263,120 C (2.6% battery) | 9.3 years |

Table 4.5: Per-node power consumption and associated expected lifetime when powered by a pair of AA batteries.

is 700 ms. These results are very similar to the very initial results presented in **watteyne16peach** indicating no degradation in performance of the SmartMesh IP network over the 3 month operation.

### Network Lifetime

Each device is powered by a pair of Energizer L-91 AA batteries. These contain a nominal 3134 mAh of charge, or 2821 mAh when accounting for a 10% decrease due to manufacturing differences. A SmartMesh IP node contains a "charge accounting" feature in which it tracks the amount of charge is has been drawing from the battery. The mote reports this number every 15 min as a field in its `HRDevice` health report. This number allows us to predict the lifetime of the device.

Table 4.5 shows charge consumed by the 16 motes inside the orchard over the 3 month period (87 days), as well as the portion of the battery this represents. Assuming the same energy consumption rate, we can extrapolate the lifetime. The node with the longest lifetime is `60-02-4b`. From Fig. 4.7, we can see that this is a leaf node. Since it does not have to relay data from any children, it is normal that this node consumes very little. The node with the shortest lifetime is `60-03-82` and has 5 years of lifetime. This shows the ultra-low power consumption of the SmartMesh IP network.

### 4.3.9 Lessons Learned

Although frost events may appear harmless, they yield enormous losses in fruit production, for example peaches. The most important lesson learned is that IoT is the right technology for this precision agriculture application, and that using it makes a huge monetary difference. Moreover, as seen throughout this project, IoT technology is ready for this type of application. We strongly believe that the combination of technologies demonstrated through the PEACH project is the right one for a large number of "Smart Agriculture" applications. We hope that the present chapter can guide end users put together the right technical solution.

The main outcome of this project is a perfectly working end-to-end low-power wireless distributed sensor system, built exclusively from commercial off-the-shelf components. Deploying such a network took only a couple of hours, and putting all the hardware and software together only a couple of days. The resulting network is 100% reliable, offers latency below 2 s, and 3+ years of battery lifetime.

In 2006, Langendoen *et al.* wrote a foundational (and "brutally honest") paper listing everything that went wrong in a precision agriculture deployment similar to the PEACH project **langendoen06murphy** This included board failure, batteries running out, sub-optimal software tracking, etc. So what went *right* this time, about a decade later? The teams in both cases are of the same caliber, so the *wrong* conclusion would be to blame/praise the people involved. What has really happened is that the field of low-power wireless has evolved substantially, and has radically changed in that decade.

In 2006, *researchers* bought boards and programmed them from scratch with academic open-source projects. In 2016, *end users* buy complete working systems, including all the hardware and software.

In 2006, the protocol stack implemented was a combination of the most promising research papers from recent years. In 2016, the protocol stack implemented is entirely standards-based, with standards matured over years in a commercial/industrial mind-set.

In 2006, every new deployment was pushing the frontier. In 2016, tens of thousands of these network have been running for years in the most critical applications.

To put it plainly, **Wireless Sensor Network/Internet of Things technology has successfully transitioned from the academic to the commercial world**.

The result is a necessary redefinition of the work of the academic community. The era of building everything from scratch is over. This is in particular true for software implementations; a small research group is simply no match for the heavy lifting full-time development teams do. In our opinion, the energy of small and agile research teams should be spent on (1) ensuring that clever ideas transition to the commercial world through standardization and

(2) be very well connected to the commercial world, for example by continuously surveying existing products/technologies. As for point (2) above, we encourage journal editors and conference program chairs to include "lessons learned from practical deployments" onto the topics of interest.

## 4.4 SnowHow

A snow-pack monitoring system in the Sierra Nevada (California). We deployed 27 devices in a $100km^2$ area to measure snow level, solar radiation and other environmental indicators, in the mountain.

### 4.4.1 Context

Between 2012 and 2015, California suffered from the highest water drought since recordings started in this state. Up to 2/3 of its water resources are coming from the Sierra Nevada snowpack. Understanding the effect of the droughts on the mountain snowpack is crucial.

Until recently, snow melting and water displacement measurements were done by hand. This manual process is costly and not frequent enough to reflect the environment changes. There was a need for a dense (i.e. in time and space) monitoring solution to capture the spacial variability of the water.

The SnowHow project started in 2009 with the goal of monitoring the Sierra Nevada snowpack without human intervention. At the time of writing, 21 low-power wireless networks are deployed in the Sierra Nevada. They measure snow level, solar radiation, soil moisture, soil temperature, air temperature and air relative humidity in different locations. The SnowHow project currently includes the American River Hydrological Observatory (ARHO) project **zhang16longterm** with 985 sensors, and the Southern Sierra project **kerkez12design** with 300 sensors **brun16sierra**

### 4.4.2 Related Work

Bogena *et al.* highlight the potential of low-power wireless for measuring soil water content variability **bogena10potential** Pohl *et al.* do a similar analysis for understanding the snow cover **pohl14potential** Rice and Bales show how embedded sensors can be used to evaluate the water content of snow **rice10embedded** and Qian et al. propose a system using short range nodes and cellular phones to capture orchard data **qian2015farm** Simoni *et al.* use wireless sensor networks to model the hydrologic response of an alpine watershed **simoni11hydrologic** Li *et al.* summarize lessons learned from deploying a wireless sensor network for soil monitoring **li14practical** Gutierrez *et al.* use low-power wireless to monitor water and automate irriga-

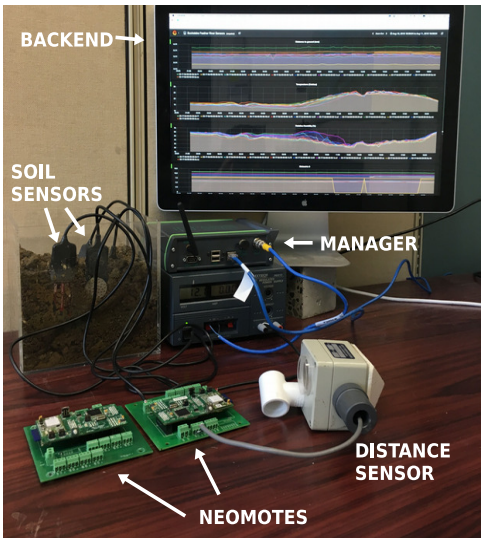



Figure 4.11: The setup of the SnowHow demonstration, including the motes, the manager, the sensors, and the web interface of the back-end system.

tion **gutierrez14automated** Moreover, Ojha et al. **ojha2015wireless** survey the state of the art of wireless sensors for agriculture.

### 4.4.3 Deployment

Each network of the ARHO and Southern Sierra deployments shares the same architecture. Sensor stations are placed in hydrologically-significant locations. Repeater nodes are added to ensure good connectivity. Sensor data is relayed to the manager node, which is connected to a Linux computer. This computer connects to the Internet through a satellite or cellular link. Seconds after the generated data is produced in the deployment site, it appears in the database and can be visualized online.

At the heart of each sensor station is a NeoMote, a low-power wireless platform commercialized by Metronome Systems, a UC Berkeley spin-off company. The NeoMote is a generic sensor platform, which features a Cypress PSoC micro-controller and a SmartMesh IP low-power wireless mote, in a hardened weather-proof design. The NeoMotes are using the SmartMesh IP technology to form low-power wireless networks with >99.999% end-to-end reliability and a over decade of battery lifetime **watteyne15industrial**

I participated in one of the AHRO deployment in the Southern Sierra Critical Zone Observatory (CZO)[16]. CZO is a program founded by the USA National Science Foundation (NSF) to study how components interact at the Earth's surface and support life. CZO allows researchers (e.g. hydrologists

[16] `https://criticalzone.org`

Figure 4.12: A cluster of 5 sensor stations in the the Southern Sierra CZO deployment. Other similar cluster make up this approximately 2 km long deployment.

and ecologists) to collaborate and improve our understanding of natural processes such as water and nutrient cycling, weathering processes, and forest function. In the Southern Sierra zone, we deployed 27 nodes in a $100km^2$ area, half of them equipped with soil sensors, to measure humidity and temperature at different levels and ultrasonic distance sensors to measure the height of the snowpack.

The deployment zone is composed of areas densely forested, with clearings. The sensor locations were selected in advance to maximize the representativity of the measurements so that they can be reused to extrapolate the data into the rest of the mountain.

Sensors include soil temperature and moisture, air temperature and humidity, wind speed, solar radiation and distance to ground.

### 4.4.4 Performance of the Network

While the deployment of this site went well, we were not able to connect it to the Internet during the time of this thesis and get the data due to a series of events: two forest fires and a broken cellular antenna. Table 4.6 depicts the network performance results obtained after 20 hours of operation.

| | |
|---|---|
| reliability | 99.97% (Arrived/Lost: 59552/15) |
| stability | 83% (Transmit/Fails: 538879/89337) |
| latency | 1000 ms |

Table 4.6: Southern Sierra CZO key network performance indicators after 20 hours.

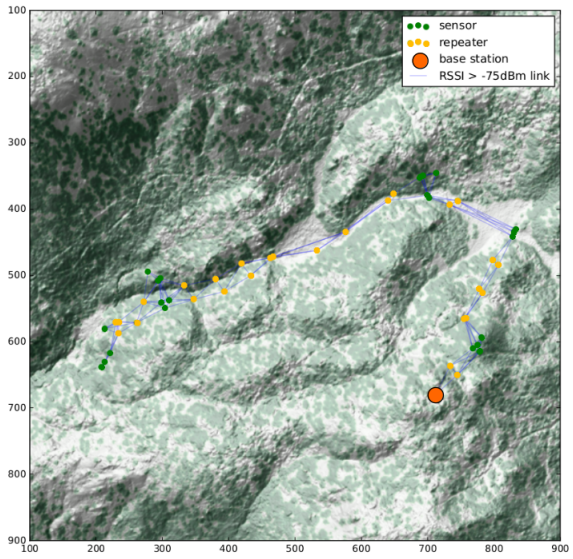



Figure 4.13: The map of the Southern Sierra CZO deployment.

### 4.4.5 Lessons Learned

While we were not able to analyze network statistics, we performed range experiments while deploying the network and were able to better understand the radio propagation in a forest environment. The main rule to follow is: "Line-of-Sight" (LoS). As the sensor location is fixed, we needed to build a network of relays to link the sensor nodes to the gateway. Our first idea was to reduce the number of hops and place the relays to form short distance paths to the gateway. We realized that building short paths through dense forest was greatly reducing the transmission range and creating links with poor quality. We finally built our network maximizing LoS links by placing relays in forest clearings and created long-but-solid paths. Those resulted guided the selection of the parameters for a machine-learning-based repeater placement algorithm **oroza17machine**

Meeting with other teams who had experience in real-world deployments also taught me things I would never learn in an office environment. Those lessons are more hardware related and anecdotal but I think deserve to be noted.

One day, one the network was not reachable anymore, as it was in summer my colleagues were able to go and investigate the situation. A massive tree felt on the manager. This risk is real and should be taken into account when building dependable systems.

Another day, the manager was showing an unusual behavior (poor link quality with its child nodes, power supply was going up and down). The wind was strong enough to bend the aluminum pole that was holding both the solar

panel and the antenna. Wind-exposed base stations are now grounded with cables.

Before deploying sensor networks into wildlife areas, I did not know bears and rodents liked to chew rubber insulated wire. All cables we deploy are now Kevlar-reinforced.

Last but not least, and there is not much you can do about it, hunters sometimes use solar panels for target practice.

## 4.5 EvaLab

A Smart Building monitoring system in Paris (France).

### 4.5.1 Context

This deployment might not be considered as a real-world deployment as it was setup to measure network performance and not to answer a real IoT application. However, we consider it as realistic as it is placed conditions that are far from ideal: external radio interferences (WiFi and Bluetooth), reflecting object (metallic ceiling), and propagation variation (people moving). We will see how we select and quantify those characteristics in Chapter 5. In this section we present the results obtained from the EvaLab agains the one obtained from the PEACH deployment.

### 4.5.2 Related Work

Smart buildings save energy by automating controls and optimizing systems. This adds value to leasing and sales of properties **king17smart** Smart Building applications face external interference, as the low-power wireless mesh networks are co-located with other technologies sharing the same frequency bands. For example, SmartMesh IP operates in the same 2.4 GHz band as WiFi and Bluetooth. A solution deployed in a building must be resilient to external interference.

Deploying IoT-based solutions in real-world application is complex, and involves elements well beyond the low-power wireless network. Barrenetxea *et al.* **barrenetxea08hitchhiker** survey experimental studies and discuss best practices from system conception to data analysis.

### 4.5.3 Deployment

The EvaLab deployment is done across a single 40 m $\times$ 20 m office building floor. We deployed the EvaLab network two times. The first time with 14 motes placed on the ceiling of one floor of the building, and 3 additional


(a) outdoors (b) indoors

Figure 4.14: Devices deployed in the Inria-Paris offices for the EvaLab deployment.

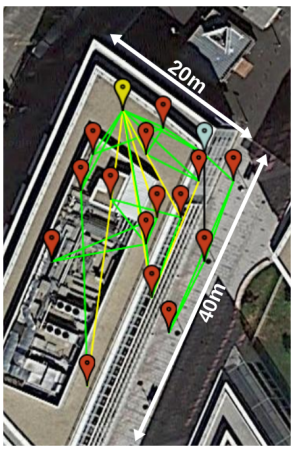
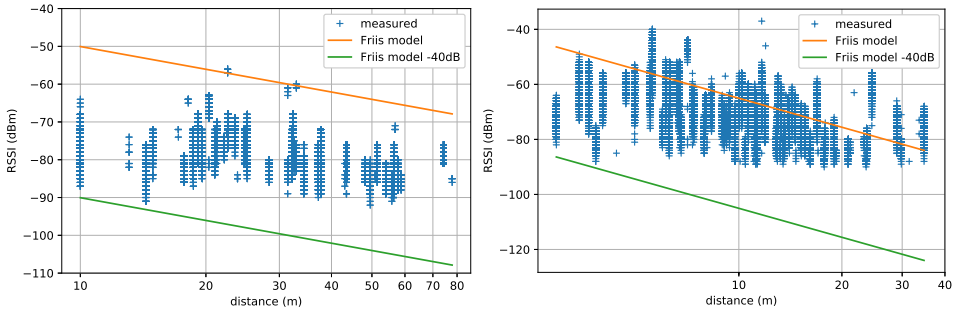



Figure 4.15: The map of the first EvaLab deployment.

motes placed right outside the building, on lamp posts (Fig. 4.15). The 14 motes inside are fixed to the ceiling using magnets; the 3 motes outside are placed in a water-tight boxes (Fig. 4.14). Thanks to the "peel-and-stick" nature of this low-power wireless technology, deployment takes less than an hour. The second time we deployed the EvaLab network, 22 were placed indoor only, also fixed on the ceiling with magnets. The study in this chapter only concern the 17 nodes deployment. We use the 22 nodes deployment in the analysis presented in Chapter 5.

About 200 people work in that building, many of them using WiFi extensively. Nodes are not attached to external sensors, each node reports temperature data every 30 s.




(a) Smart Agriculture (b) Smart Building

Figure 4.16: We observe that the RSSI measurements are roughly located between the Friis model and the Friis model shifted by −40 dB.

### 4.5.4 Intuitive Results

As we did for the PEACH results, we start by looking at the physical-layer metrics and then study the key performance estimators of the networks: end-to-end reliability and network lifetime.

#### RSSI vs. Distance

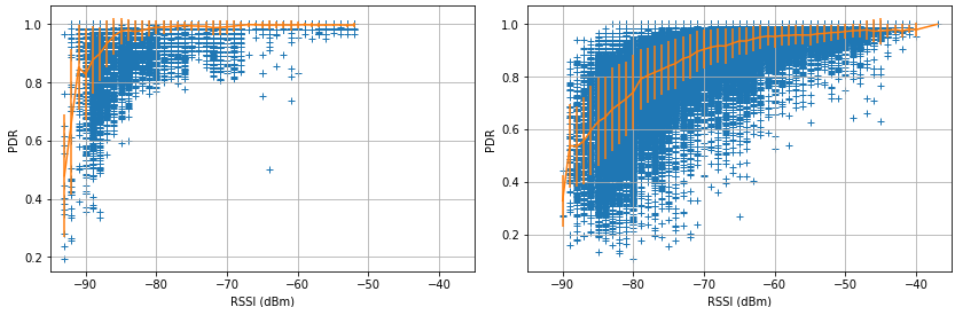
The Friis transmission model **saunders07antennas** gives the relationship between the Received Signal Strength (RSSI)[17] in free space. While the Friis transmission model does *not* apply directly to our real-world deployment[18], we observe in Fig. 4.16 that the individual RSSI values are located between the Friis model, and the Friis model offset by −40 dB. This corroborates the results from **zats10wireless** Variability is lower on the Smart Agriculture deployment, given that there is almost no environmental change, compared to what happens on the Smart Building deployment where people move around creating rapid fading changes.

#### Wireless Waterfall

Due to the inherent physical unreliability of the radio medium, it is impossible to know if a future transmission will succeed or not. The Packet Delivery Ratio (PDR) is the portion of successful link-layer transmissions over the total number of link-layer transmission attempts. A failed attempt means that the link-layer frame needs to be re-transmitted; this does *not* mean the packet is lost. Over a period of 3 months, 140,897 `HRNeighbors` messages are collected in the Smart Agriculture deployment and 128,072 in the Smart Build-

[17] Strictly speaking, the RSSI is the Received Signal Strength *Indicator*, a value returned by the radio chip. Because of its prevalence in low-power wireless literature, we use RSS and RSSI interchangeably.

[18] a Log-Normal model would be more appropriate for indoor environment




(a) Smart Agriculture (b) Smart Building

Figure 4.17: The PDR/RSSI "waterfall" plot. The Smart Building plot is shifted right compared to the Smart Agriculture plot, indicating an environment where external interference is present.

ing deployment. These contain, for a given node, the number of link-layer transmission attempts and successes to each of its neighbors. We remove the portion of neighbors with no transmission and keep only the DC9003 motes, resulting in a total of 69,643 messages (approx. 49% from the total number of `HRNeighbors`) for the Smart Agriculture deployment and a total of 93,135 messages (approx. 73%) for the Smart Building deployment.

Fig. 4.17 plots the PDR and the RSSI of these 69,643 and 93,135 messages. For readability, we also plot the average/deviation of the data for a given RSSI value. Because of its shape, we name it the "waterfall plot".

For the Smart Agriculture deployment, the average PDR of the links is very good (>95%) above -85 dBm. Below that value, the PDR rapidly degrades, indicating that, on these links, frequent retransmissions happen. For the Smart Building deployment, the PDR starts to degrade at -60 dBm. Note that, if the network were using a non-schedule MAC layer (e.g. ZigBee), the PDR for the same RSSI would be lower than in Fig. 4.17 because of collisions. The device manufacturer documentation **smip_app_note** indicates that a path is considered as "bad" when:

- RSSI>-80 dBm and PDR< 50%
- RSSI>-70 dBm and PDR< 70%

This is not the case in any of the two deployments.

The waterfall plot allows us to assess the level of external interference. In the presence of external interference, the waterfall plot is either shifted to the right with very few paths below -70 dBm, or does no constantly increase with RSSI.

The waterfall plot in the Smart Agriculture deployment (Fig. 4.17a) is "clean", meaning that the SmartMesh IP network is not experiencing high levels of external interference from co-located wireless devices. Especially

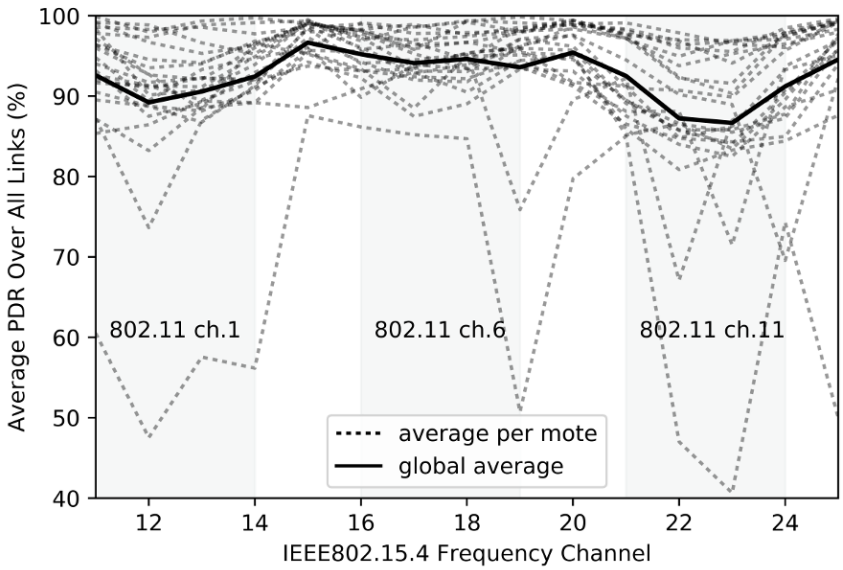



Figure 4.18: In the Smart Building deployment, external interference from nearby WiFi devices causes some retries (the PDR is lower) *without* impacting end-to-end reliability which stays at 100% (see Section 4.5.4).

when comparing both, the waterfall plot in the Smart Building deployment *is* shifted to the right (Fig. 4.17a), indicating the SmartMesh IP network is experiencing external interference. This is expected, as several hundred WiFi devices and tens of WiFi access points are operating in the building on IEEE802.11 2.4 GHz channels 1, 6 and 11.

A fourth type of Health Report allows us to measure the link quality per channel. The `HRExtended` are generated by each mote every 20 minutes and indicate the number of transmission attempts and link-layer retransmissions *for each frequency*, allowing us to calculate the PDR per channel for each mote. Fig. 4.18 shows the average PDR collected over a 24-hour period on a business day. During that period, 1402 `HRExtended` were collected. We can clearly observe the PDR drop of the IEEE802.15.4 links that share the same channel bands as IEEE802.11.

Yet, despite the high external interference, the Smart Building deployment exhibits 100% end-to-end reliability (as detailed in Section 4.5.4), underlying the resiliency of SmartMesh IP to external interference.

**End-to-End Reliability**

We expect the SmartMesh IP network to offer wire-like reliability. Table 4.7 confirms that this is the case. It presents statistics gathered over the 15-

| | Smart Agriculture | Smart Building |
|---|---|---|
| reliability (Arrived/Lost) | **100%** (693,844 / 0) | **100%** (431,193 / 0) |
| average PDR (Transmit/Fails) | 95% (4,405,569 / 258,778) | 87% (19,807,535 / 2,488,149) |
| latency | 700 msec | *not measured.* |

Table 4.7: The overall network performance in the 15-25 July 2016 period (Smart Agriculture) and the 12 Nov. 2016 - 12 Feb. 2017 period (Smart Building).

25 July 2016 period in the Smart Agriculture deployment, and over the 12 November 2016 - 12 February 2017 period in the Smart Building deployment. Both sensor data and network statistics are taken into account.

It shows that, as none of the 693,844 and 431,193 packets generated in the networks was lost, the end-to-end reliability is 100%. The average PDR over all links is very high (95% and 87%), indicating that the nodes are deployed close enough to one another. Finally, the average latency over all nodes is 700 ms for the Smart Agriculture deployment. These results are very similar to the very initial results presented in **watteyne16peach** indicating *no* degradation in performance of the SmartMesh IP network over the 3 month periods.

**Network Lifetime**

Each device is powered by a pair of Energizer L-91 AA batteries[19]. These contain a nominal 3134 mAh of charge, or 2821 mAh when accounting for a 10% decrease due to manufacturing differences. A SmartMesh IP node contains a "charge accounting" feature in which it tracks the amount of charge it has been drawing from the battery. Each mote reports this number every 15 min as a field in its `HRDevice` health report. This number allows us to predict the lifetime of the device.

Table 4.8 shows charge consumed by the motes over the two 3 month periods. Assuming a constant energy consumption rate, we extrapolate the lifetime. The nodes with the longest lifetime (8 years) are all leaf nodes as we can see from Fig. 4.15. Since they do not have to relay data from any children, it is expected for these motes to consume the least. The mote with the shortest lifetime is `30-60-ef`, with a 4 year battery lifetime. This is expected, as this mote relays an important amount of data. This confirms the ultra-low power consumption nature of the SmartMesh IP network.

[19] http://data.energizer.com/pdfs/l91.pdf

| MAC | charge consumed* | lifetime |
|---|---|---|
| 30-60-ef | 695 C | 4 years |
| 38-0f-66 | 461 C | 6 years |
| 3f-f8-20 | 380 C | 8 years |
| 3f-fe-87 | 549 C | 5 years |
| 3f-fe-88 | 718 C | 4 years |
| 58-32-36 | 311 C | 7 years |
| 60-01-f8 | 387 C | 8 years |
| 60-02-1b | 371 C | 8 years |
| 60-02-4b | 406 C | 7 years |
| 60-03-82 | 395 C | 8 years |
| 60-05-5f | 386 C | 8 years |
| 60-05-69 | 509 C | 6 years |
| 60-05-78 | 364 C | 8 years |
| 60-05-ab | 381 C | 8 years |
| 60-06-27 | 422 C | 7 years |
| 60-08-d5 | 432 C | 7 years |

(a) Smart Agriculture

| MAC | charge consumed* | lifetime |
|---|---|---|
| 38-03-dd | 459 C | 5 years |
| 58-e9-ca | 411 C | 6 years |
| 58-e9-cb | 407 C | 6 years |
| 58-eb-5b | 423 C | 6 years |
| 58-eb-64 | 322 C | 8 years |
| 58-eb-67 | 468 C | 5 years |
| 58-eb-69 | 243 C | 7 years |
| 58-f3-17 | 357 C | 7 years |
| 58-f4-f8 | 402 C | 6 years |
| 58-f5-23 | 416 C | 6 years |
| 58-f5-3c | 412 C | 6 years |
| 58-f5-58 | 387 C | 6 years |
| 58-f8-63 | 198 C | 9 years |
| 58-f8-78 | 233 C | 7 years |
| 58-f8-8f | 439 C | 6 years |
| 58-f9-c4 | 325 C | 8 years |

(b) Smart Building

* over the 3.5 month and 3 month periods, respectively.

Table 4.8: Per-node charge consumed and associated expected lifetime when powered by a pair of AA batteries.

### 4.5.5 Not so Intuitive Results

Results from Section 4.5.4 are "intuitive" in that they corroborate previous measurements **watteyne16peach** or confirm theoretical/lab results **watteyne10mitigating**, **watteyne09reliability**, **watteyne15industrial** This section presents results which we believe go against popular belief. This classification is necessarily subjective.

We first show that links are, in fact, symmetrical. We then show that, through the use of TSCH, the low-power wireless topology is, in fact, extremely stable.

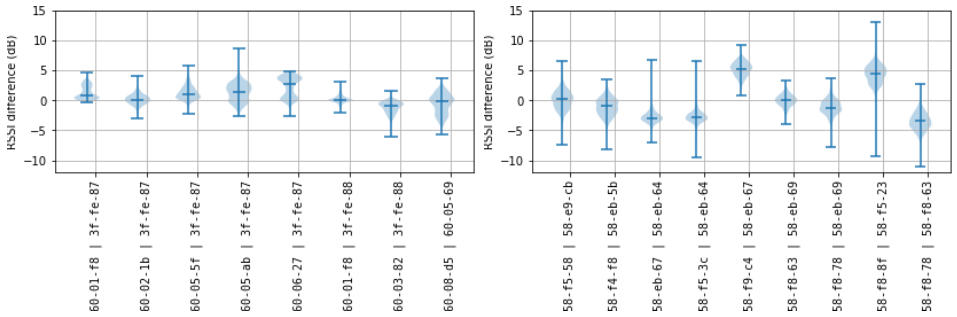
#### Link (A)Symmetry

Motes report the average RSSI value of the packets received from each neighbor in their `HRNeighbors` health reports. Because the network uses channel hopping, the reported RSSI values are also averaged over 15 IEEE802.15.4 frequencies **std_ieee802154_2015** In this section, we use the term "RSSI" to denote the average RSSI over 15 frequencies.

A common assumption is that links between neighbor low-power wireless devices are hugely asymmetric. That is, on a link between nodes $A$ and $B$, $A$ receives $B$'s link-layer frames with an RSSI very different from the frames $B$ receives from $A$. Numerous routing protocols (often standardized **rfc3626**) reuse that assumption and start with a costly step of filtering out asymmetric links.

We look at the link statistics between 18 June 2016 and 4 July 2016 (16 days) in the Smart Agriculture deployment, and between the 12 November 2016 and 7 February 2017 (87 days) in the Smart Building deployment. The dataset contains 411,132 `HRNeighbors` messages received from 14 DC9003 nodes (same hardware). During that period, 21 links are active with at least 250 transmissions for each link. For each of these links, we compute the difference between the average RSSI in each direction. Results are presented in Fig. 4.19.

In 99.6% of the cases, the difference does not exceed 7 dB. Looking at Fig. 4.17, this translates into only a handful of percentage points difference in PDR. This means the links can be considered symmetric. This result is in-line with the physical phenomenon that the signal traveling from $A$ to $B$ undergoes the same attenuation as that from $B$ to $A$. This result would *not* hold if the neighbor radios had a different transmit power or sensitivity. That being said, discussions on link (a)symmetry at the routing layer is largely artificial, as virtually all state-of-the-art medium access control (MAC) protocols uses link-layer acknowledgments, thereby naturally filtering out asymmetric links.




(a) Smart Agriculture (b) Smart Building

Figure 4.19: The difference in RSSI between the two directions of the wireless links with the highest number of exchanged messages. The violin plots show the distribution of the value and the standard deviation.

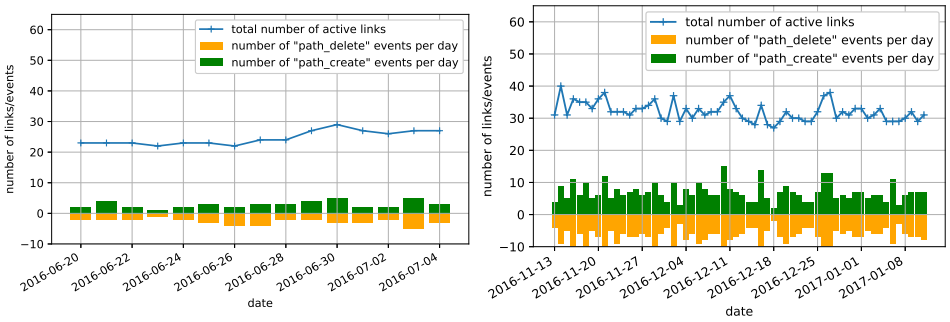
### Network Stability

Wireless is unreliable in nature. It is normal that some wireless links interconnecting motes "come and go". That is, links that have been performing well (e.g. PDR>90%) can suddenly disappear (e.g. PDR<10%). Similarly, nodes that were not able to communicate can suddenly hear one another perfectly.

The question, however, is what time scale is considered. Early academic work on low-power wireless **srinivasan08beta** has looked at the "burstiness" of the wireless links, i.e. changes over the course of 10-1000's ms. Some follow-up work has taken the assumption that wireless links are so unstable that only a reactive routing approach works. In this section, we infirm this statement by looking at the stability of the network.

In particular, we look at the `path_delete` and `path_create` events. These are generated each time a node adds/deletes a neighbor to communicate with, which happens for example when the routing topology changes. The number of `path_delete` and `path_create` events is a direct measurement of network stability. We remove the nodes that do not respect the deployment requirement of having at least two parents to associate with (we remove one node in the Smart Agriculture deployment and three nodes in the Smart Building one). Due to the lack of second parent, these nodes were producing over 20 times the amount of messages than all the other nodes combined.

Fig. 4.20 shows the number of `path_delete` and `path_create` events per day, over the 16 day (Smart Agriculture) and 87 day (Smart Building) periods. For reference, the total number of links in the network is also depicted. There are less than 5 `path_delete` or `path_create` events *per day* in the entire Smart Agriculture network, and at most 15 in the Smart Building network. This means that links, once established, remain useful for days/weeks at a time, and that the network is extremely stable. We attribute the higher churn in the Smart Building deployment to the presence of significant external




(a) Smart Agriculture

(b) Smart Building

Figure 4.20: Network stability: the number of `path_create` and `path_delete` events generated per day over a 16 day (Smart Agriculture) and two month (Smart Building) period. The top line shows the total number of active links.

interference.

This stability can largely be attributed to the use of channel hopping. Changing frequency for each frame is known to efficiently combat multi-path fading and external interference **watteyne09reliability** the two major causes of instability. If channel hopping were not used, selecting links with high PDR would not be sufficient as they could be affected by external interference at any time and become unstable. It does not contradict the findings of **srinivasan08beta** it just means that link-layer retransmissions can efficiently cope with link burstiness, and that the multi-hop topology can remain very stable.

### 4.5.6 Conclusion and Lessons Learnt

We analyzed the network statistics generated by two low-power wireless mesh networks deployed in real-world conditions. The first network is deployed in a peach orchard in Argentina, in a Smart Agriculture scenario. Its 21 motes have produced 369,276 statistic measurements over the course of 3.5 months. The second network is deployed in an office building in Paris, in a Smart Building scenario. Its 17 motes have produced 386,929 statistic measurements over the course of 3 months.

We use a "waterfall" plot to show that the two networks are subject to different amounts of external interference from other wireless devices deployed in the same area. The SmartMesh IP network delivers its exceptional performance, with 0 packets lost out of 693,844 (Smart Agriculture) and 431,193 (Smart Building) received (100% reliable) and 4-8 years of battery lifetime on a pair of commercial AA batteries. This is representative of the performance of 6TiSCH technology.

While it is often assumed that wireless links are asymmetric, we show

to the contrary that the difference in RSSI averaged over 15 IEEE802.15.4 channels does not exceed a handful of dB. We show that the network is extremely stable, with less than 5 links being added or deleted per day in the Smart Agriculture deployment, at most 15 in the Smart Building deployment. We attribute this performance to the use of Time Synchronized Channel Hopping (TSCH) technology at the heart of the SmartMesh IP products.

We conclude that SmartMesh IP is a perfectly suitable IoT solution for Smart Agriculture and Smart Building applications.

## 4.6 SmartMarina

A metering and management solution for marinas, deployed in Agde (France).

### 4.6.1 Context

Marinas are quickly evolving from sailing spots to floating neighborhoods. It is now common for people to live on their boat year-round, and for boats to be rented for just a week-end through online platforms. Today, living or staying on a boat is often cheaper that buying or renting an apartment. Similarly, in coastal areas, the marina is often the center of the city, so an ideal location for lodging. As a result, the trend is not going to end any time soon. Today's marinas are tomorrow's smart cities. And as the marina is evolving, so are the needs.

From a marina management point of view, automatic mooring management and electricity/water monitoring allows personnel to free up to welcome visitors and focus entirely on their well-being. Year-road boat owners and occasional marina visitors now can enjoy new services, from increased mooring availability to remote monitoring and alerts about the state of their boat.

The combination of embedded micro-controllers, low-power wireless communication and sensors/actuators offers tremendous opportunities for marinas. Off-the-shelf "Internet of Things" technology can now be used to detect the presence of boats in moorings, track usage of water and electricity on a per-boat basis, track a boat in real-time as it enters the marina, etc. Because no wires need to be installed -– neither for power, nor for communication –- installation can be done in a matter of hours in a peal-and-stick fashion. Pontoons can be moved, rearranged or removed, without having to worry about the smart devices mounted on it.

The goal of the SmartMarina project is to build a system composed of sensors deployed all over the marina, and advanced software to monitor the occupation of moorings, and the electricity and water consumption on each


Figure 4.21: A map of the SmartMarina deployment.

spot. The result is a system that allows more efficient management and new services.

Today's boat owners can install a private system on their boat and receive an alert when something suspicious happens, such as a break-in, a fire or a leak. But since the owner are most often away from the boat, it is the marina which needs react. The goal of the SmartMarina project is to place the marina at the center of the solution.

### 4.6.2 Related Work

Using IoT for marine-based applications is a recently growing field. One application is boat location detection for management or border intrusion **felemban2013advanced**, **arifin17iotbased** To the best of our knowledge, no system exist that takes into account the new trend in the marinas and consider them as floating cities.

### 4.6.3 Deployment

The 19-node SmartMarina deployment is done as part of a "smart parking for boats" project at the Cap d'Agde marina, in Southern France[20]. Nodes are attached to ultrasonic sensors to detect the presence of boats on the different moorings. The network is deployed along a 50-boat pier.

External interferences are present as WiFi is deployed across the marina and used extensively by boat owners.

[20] `http://www.smartmarina.org/`

### 4.6.4 Results

Table 4.9 lists the key network performance indicators after 5 months of operation. We can see that high reliability of SmartMesh IP is confirmed despite having a low 70% average link stability.

| | |
|---|---|
| reliability | 99.999% (Arrived/Lost: 3365708/14) |
| stability | 69% (Transmit/Fails: 22515224/6939652) |
| latency | 1400 ms |

Table 4.9: The SmartMarina network performance indicators after 5 months of operation.

### 4.6.5 Lessons Learned

Deploying a 19-node WSN in a marina took us (a team of 3) less than 3 hours. We confirm that TSCH-based networks can offer high reliability even in a marina environment with large objects (i.e. boats) moving around.

## 4.7 Conclusion

### 4.7.1 Summary

Table 4.10 summarizes the real-world deployments we worked on and their corresponding environment.

| Deployment | #nodes | area ($m^2$) | Environ. | Multipath | External Interf. |
|---|---|---|---|---|---|
| PEACH | 23 | $11km^2$ | outdoor | No | No |
| SnowHow | 27 | $100km^2$ | outdoor | No | No |
| EvaLab | 17 & 22 | $800m^2$ | indoor | Yes | Yes |
| SmartMarina | 19 | $5km^2$ | outdoor | Yes | Yes |

Table 4.10: A summary of the real-world deployments we worked on.

### 4.7.2 Lessons Learned

I provide here a non exhaustive list of *wrong* assumptions I had before deploying real-world networks.

**Deployment Related**

- it is always easy to move a mote
- it is always easy to add a mote

- a tree will not fall on the base station
- a bear will not eat your wires (rodents neither)
- the wind will never bend an aluminum pole
- a better antenna gain means a better range
- two nodes side by side will always hear each-other
- RSSI indication is sufficient to estimate a link quality
- firmware testing is optional
- you have unlimited cellular data plan
- a cellular access is always available
- radio communication always passes through the windows
- radio communication always passes through a building floor

**Network Related**

- a sphere-based propagation model is valid
- the links quality are always asymmetric
- the L3 network paths are very unstable
- reducing a node's publish-rate below 30 s saves energy (The nodes will send empty messages to resynchronize any way. The time depends on the clock drift.)

### 4.7.3 Challenges & Contributions

In this chapter I presented the challenges we faced when deploying networks in real-world scenarios. Those challenges are mainly deployment-related and can be considered as non-relevant from a research point of view. I disagree with that statement and argue that applicability should be taken more into account when designing network protocols. I gathered practical insight from issues I faced when I deployed devices in rough environments. I believe I would not have fully understood those insights without deploying those networks. **My contribution is to describe those insights so that unrealistic assumptions can be identified more easily.**

I analyzed the data obtained from those deployments and presented simple performances statistics. While those simple results do not highlight new scientific issues, I believe they are valuable in the sense that they provide a practical reminder of what performances are currently achieved by off-the-shelf products. Those results can act as a reference and help the research community focusing on problems that are not already solved.

I went further and analyzed the symmetry of the radio links of those networks. I showed that, given certain conditions, radio links can be considered

symmetric. I also looked at the network paths stability and showed that, in some of our networks, paths are very stable.

Finally, to the best of my knowledge, there are no TSCH-based network connectivity and statistic traces available to the research community. Thank to those deployments we now have large datasets on top of which we can study. I believe I gathered the largest real-world TSCH-based dataset available to the research community.

# Chapter 5

# Characterizing Networks

**In this chapter, we explain how we gathered data from IoT testbed deployments and compared them with real-world datasets. We identify the key metrics and behaviors of the network that we must take into account when developing a protocol for the IIoT. We look at external interferences, multipath fading, and their variations over time.**

We gathered large network statistic datasets and understood how networks behave in real-world environments. However, the datasets we collected are not dense enough in time and radio frequency to fully grasp what happens at the MAC level. The SmartMesh IP technology we used in our real-world deployment produces one full set of statistics per node every 15 min, and each statistic thus contains an aggregation of the counters and network configurations produced since the previous statistic was sent. To really understand what happens at the MAC level, we need to know the precise status of the MAC layer resources at any given time. That includes, for instance, how many cells are allocated, how many retransmissions are needed to successfully transmit a frame, or what contributes most to packet latency. Getting enough data to answer those questions using a real-world network is impossible as the amount of data we need to collect goes beyond the network traffic capacity. We thus use deployments dedicated for network data collections, where each node can communicate both wirelessly – for standard WSN operation –, and through wires – for debugging and logging information. Those network are called *testbeds*.

## 5.1 Introduction

Designing a low-power wireless networking protocol typically involves early back-of-the-envelope analysis, high-level simulation, and experimental evaluation to benchmark its performance. After that, the protocol is considered ready to be deployed in real-world applications. *But is it really?*

*How can one be sure the conditions in the testbed were varied enough that the protocol was actually tested in real-world conditions? How realistic is a testbed deployment compared to real-world scenarios?*

We focus on connectivity (the characteristics of the wireless links between nodes) of IEEE802.15.4-based low-power wireless networks and want to (1) compare the connectivity between testbeds and real-world deployments, and (2) propose a methodology to verify that the testbed evaluation includes all key connectivity characteristics seen in the real world. The goal of this methodology is to ensure that a protocol that performs well on a testbed also does so when moving beyond testbeds.

The methodology we adopt is the following. We start by gathering a large set of connectivity traces on testbeds. Then, we extract from the previous real-world deployment traces the three main connectivity characteristics: presence of external interference, level of multi-path fading, and amount of dynamics in the environment. In the process, we show that some testbeds do not feature all three characteristics. Finally, we propose a methodology for ensuring testbed results are realistic and describe the associated tools.

This chapter makes the following contributions:

1. A methodology to collect dense connectivity datasets.
2. Mercator: a tool for collecting dense connectivity datasets in testbeds: Mercator is fully described in this chapter and the related code is published under an open-source license.
3. Eleven connectivity datasets available to the research community, from both testbeds and real-world deployments, containing 2,873,156 Packet Delivery Ratio (PDR) measurements gathered over a cumulative 170,037 mote-hours of operation.
4. A check-list to asses the realism of a (testbed) deployment.
5. The visual "waterfall plot" tool to instantaneously evaluate connectivity characteristics.

## 5.2 Related Work

This chapter focuses on the evaluation of protocol proposals. In particular, on making sure that a protocol that has proven to perform well in a testbed, also does so when used in real-world applications.

New protocol proposals typically are evaluated through analysis, simulation, and experimentation. Experimentation is done mostly on testbeds: permanent instrumented deployments focused entirely on protocol benchmarking. There is no shortage of open-access testbeds, including **Indriya doddavenkatappa12indriya IoT-lab adjih15iotlab** and **Tutornet gomes16insights** Typical testbeds consist of between 50 and 400 nodes deployed in an indoor environment, usually a university laboratory. **Tonneau** *et al.* put together an up-to-date survey of over a dozen open-access testbeds **tonneau15choose**

**Tala** *et al.* list the main characteristics to look at when performing an experiment on a testbed, including transmission power, type of hardware, and hardware acceleration **tala12guidelines**

Since each testbed is different, it is important to understand the connectivity between nodes in a particular site. It is equally important to make sure that this connectivity has the same key characteristics as real-world deployments.

**Papadopoulos** *et al.* study the connectivity in the IoT-LAB Strasbourg testbed, and show how the shape/structure of the building, WiFi interference, and time of day impact experimental results **papadopoulos16importance**

**Watteyne** *et al.* perform a similar analysis on IoT-LAB Grenoble, a 350-node testbed deployed in a 65 m $\times$ 30 m office building **watteyne15lessons** Each node transmits 100-frame bursts while all others listen and record received frames. This process is repeated for all 16 IEEE802.15.4 channels at 2.4 GHz. The authors quantify multi-path fading, and show that WiFi beaconing significantly impacts network performance.

With such variety of testbeds, being able to conduct reproducible experiment becomes important. **Papadopoulos** *et al.* show that only 16.5% of the studied experimental-based work propose reproducible results **papadopoulos16importance**

Somewhat more fundamentally, it is of paramount importance to ensure that a solution evaluated on the testbeds "looks like" real-world deployments. If that is not the case, a solution might work perfectly on a testbed, but fail when deployed beyond testbeds.

**Zhao** *et al.* did some early work on measuring the connectivity between nodes deployed in realistic environments **zhao03understanding** They deployed 60 nodes in a building, a forest and a parking lot.

More recently, **Dong** *et al.* proposed a methodology for collecting data traffic and analyzing the impact of packet delivery ratio in different protocols **dong14measurement** The data is collected from a real-world deployment in the wild, with 343 nodes deployed for 10 days. All nodes generate three types of packets every 10 minutes, containing raw application data, link quality, and routing statistics. They propose a detailed algorithm to analyze the root causes of network loss, and account for issues related to the environment, as well as for software and hardware problems. The proposed methodology is not easily extensible to collection and analysis of other applications, since

statistic generation is protocol-dependent. Even though the experiment was performed in a large-scale deployment, the network runs on a single channel, and from the results, it is clear that the links were very stable and not influenced by external interference.

**Doherty** *et al.* deployed a 44-node low-power wireless network in a 76 m × 69 m industrial printing facility for 26 days **doherty07channel** Authors show that the PDR varies over frequency (i.e. because of multi-path fading) and time (i.e. because of dynamics in the environment). The authors propose a simple model that takes multi-path variation into account.

**CRAWDAD**[1] is a community archive that gathers wireless traces, including on connectivity, since 2002 **kotz09crawdad** It is a web platform to store and document traces of production wireless networks. To date, the platform has 121 datasets on different application and technologies. For instance, the dataset used in **fu15experimental** is available. This dataset is the results of the analysis of different IEEE802.15.4 parameters for a network deployed in an indoor environment for 6 months. *As an online addition to this thesis, the 11 datasets gathered (see Section 5.3.3) are available on the CRAWDAD platform.*

With such datasets available, some run simulations on them, i.e. replacing the propagation model at the PHY layer of simulators.

**Watteyne** *et al.* analyze multi-channel networks based on frequency hopping **watteyne09reliability** The authors deploy 46 TelosB nodes in a 50 m × 50 m office environment. They show that end-to-end reliability can be improved through channel hopping. The results are based on simulations that take into account the connectivity datasets (more precisely, the Packet Delivery Ratio) obtained from a deployment in a working office. Even though the datasets utilized are realistic, the chosen environment is very limited in size, and the results may not be applicable to other scenarios, such as large-scale and/or outdoor deployments.

We make two main observations from surveying related work. First, only very few connectivity traces are gathered on testbeds, and their connectivity is not studied well. Most often, protocols are being evaluated, without really knowing whether the connectivity in the testbed resembles that in real-world scenarios. Very little is done in related work to show the completeness of the evaluation, i.e. demonstrate that the testbed(s) used for evaluation contains the same connectivity characteristics as real-world deployments. Second, very little has been done to verify that the connectivity in these testbeds resembles real-world deployment connectivity. The impact of this second point is particularly important, as, without it, one cannot really trust that a solution that works on a testbed will also work in a real-world deployment.

This chapter contributes to answering that need in several steps. First, we present Mercator, the tools we developed and used to collect connectivity

[1] `https://crawdad.org/`

traces which are dense in time, space and frequency (Section 5.3). Second, we present the 11 datasets collected, containing 2,873,156 measurements, both on testbeds and real-world deployments (Section 5.3.3). These datasets are made available to the community, and to the best of our knowledge, are the largest dataset dense in time, frequency, and space available to date. Third, we make a number of observations, highlighting, in particular, the fact that testbeds can easily not present a key connectivity characteristic (Section 5.5). This means that there is a real risk of having a protocol that performs well on a testbed, but fails in real life. Lastly, we present a 5-point checklist to verify that a testbed features the key connectivity characteristics of real-world deployments (Section 5.6).

## 5.3 Mercator: Dense Connectivity Datasets

This section details how we collected the connectivity datasets that we will then analyze in Section 5.5. Section 5.3.1 starts by describing the methodology, and introduces the necessary terminology. Section 5.3.3 introduces Mercator, the tool used to collect connectivity datasets on testbeds. Section 5.3.4 details the 3 testbeds, 3 long-term real-world deployments, and 1 short-term real-world deployment done to collect the datasets. Section 5.4 lists the 11 datasets collected, containing 2,873,156 PDR measurements gathered over a cumulated 170,037 mote-hours of operation.

### 5.3.1 Methodology and Terminology

Our goal is to gather *dense* connectivity datasets and learn lessons from them.

We are interested in the connectivity between the nodes in an IEEE802.15.4-based low-power wireless network, and quantify the "quality" of its links using their Packet Delivery Ratio (PDR). We operate at 2.4 GHz, the most commonly used IEEE802.15.4 frequency band. The PDR of a link between nodes $A$ and $B$ can be measured as the ratio between the number of link-layer acknowledgments frames received by node $A$, and the number of link-layer frames sent from node $A$ to node $B$. A link with PDR $= 50\%$ means that, on average, node $A$ has to transmit the same frame twice to node $B$ to receive an acknowledgment and consider the communication successful. We consider the PDR of a link to be a good indicator of the "quality" of a link, and prefer it over other indicators such as the Received Signal Strength Indicator (RSSI), which are related but hardware-dependent.

We call **PDR measurement** the measurement of the PDR (a number between 0% and 100%) between two nodes, at a particular time. A **dataset** consists of all the PDR measurements collected during one experiment.

We want the dataset to be dense along 3 axes:

1. dense in *time*, as we want to analyze the PDR of a link evolving over time,
2. dense in *frequency*, as we want to see the impact of the communication frequency used on the PDR at a particular time,
3. dense in *space*, i.e. collected over as many environments as possible to draw conclusions that apply to various use cases.

Datasets are collected on both testbeds and real-world deployments. While the data contained in the datasets are equivalent (and can be compared), the hardware and tools in both cases are different. We hence use two tools (Mercator and SolSystem), both creating equivalent data.

### 5.3.2 IoT-LAB

The IoT-LAB **adjih15iotlab** is a 2728-node testbed platform that allows to run IoT experiments on real hardware. The project is part of the FIT (Future Internet of Things) Consortium, an "open large-scale testing infrastructure for systems and applications on wireless and sensor communications" [2]. At the time of writing, the IoT-LAB features over 2000 wireless sensor nodes spread across 6 different sites in France. The number of devices per site varies between 29 and 640. Devices can be fixed or mobile.

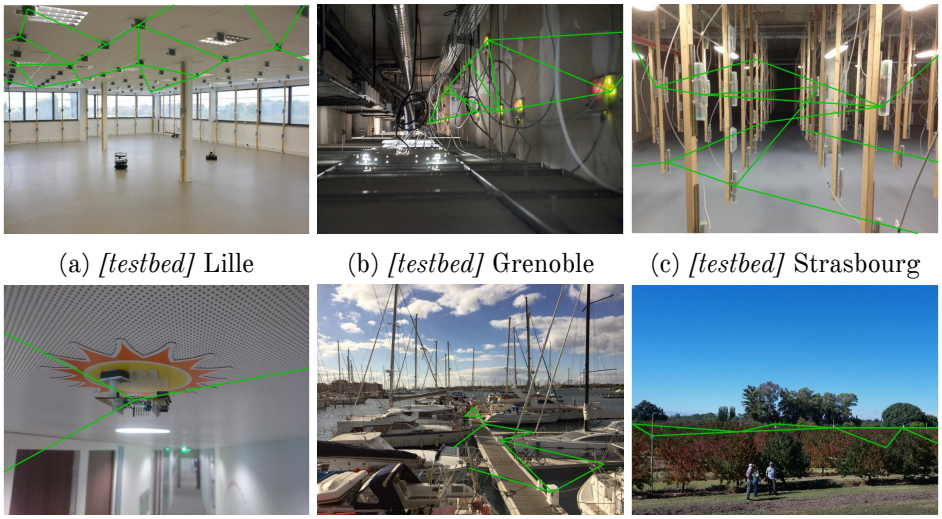
### 5.3.3 Mercator: Testbed Datasets

The 3 testbeds we use offer the ability to load arbitrary firmware directly on IEEE802.15.4-compliant nodes. These nodes are deployed in a particular location (detailed in Section 5.3.4), and while our firmware executes, we have access to the serial port of each device. This means we are able to (1) receive notifications from the nodes, and (2) send commands to the nodes, without interfering with the radio environment.

We developed Mercator, a combination of firmware and software specifically designed to collect connectivity datasets in testbeds[3]. The same firmware runs on each node in the testbed; the software runs on a computer connected to the testbed, and drives the experiment. The firmware allows the software to control the radio of the node, by sending commands to its serial port. The software can send a command to a node to either transmit a frame (specifying the frequency to transmit on), or switch the remote node to receive mode (on a particular frequency). In receive mode, the node issues a notification to the software each time it receives a frame.

[2] `https://www.fit-equipex.fr/`

[3] The Mercator source code is published under a BSD open-source license at `https://github.com/openwsn-berkeley/mercator`.



(a) *[testbed]* Lille (b) *[testbed]* Grenoble (c) *[testbed]* Strasbourg

(d) *[real-world]* EvaLab and Inria-C (e) *[real-world]* SmartMarina (f) *[real-world]* PEACH

Figure 5.1: Pictures of the testbeds and real-world deployments we collect dense connectivity datasets in. Green lines are added to suggest wireless communication between nodes. They show the position of the nodes, but do not per-se represent the exact connectivity collected in the datasets.

All frames are 100 B long, and contain the necessary fields (unique numbers, addressing fields, etc.) to filter out possible IEEE802.15.4 frames sent by nodes outside the experiment.

At the beginning of an experiment, the same firmware is loaded on all nodes. The software is responsible for orchestrating the experiment, which has a pre-set duration. The software starts by having a particular node transmit a burst of 100 frames, on a particular frequency, while all other nodes are listening to that frequency. By computing the portion of frames received, each listening node measures the PDR to the transmitting node, at that time, on that frequency. The PDR ranges from 100% if the node received all frames, and 0% if it received none. The software repeats this over all 16 available frequencies, and all nodes, in a round-robin fashion, until the end of the experiment. The dataset resulting from the experiment contains the PDR measured over all source-destination pairs, all frequencies, and throughout the duration of the experiment.

Mercator has been used on 3 testbeds (Section 5.3.4), resulting in 5 datasets (Section 5.4).

### 5.3.4 Deployments

Fig. 5.1 shows pictures of the 7 deployments used to generate the datasets.

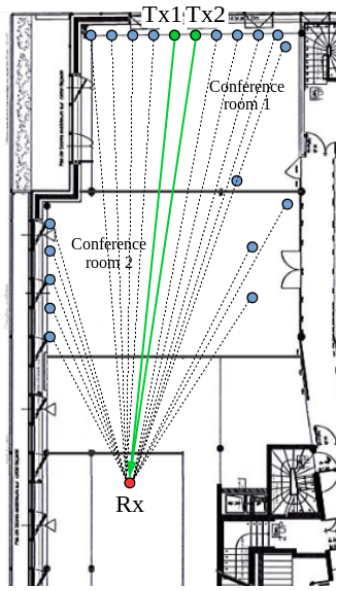



Figure 5.2: Floor plan of the Inria-C deployment.

We run Mercator on the Lille, Grenoble and Strasbourg IoT-LAB sites. On the Lille site (Fig. 5.1a), nodes are deployed on the ceiling and walls of a single large, mostly empty, room in an office building. On the Grenoble site (Fig. 5.1b), nodes are deployed along four interconnected corridors of an office building, hidden between the dropped ceiling and the roof. On the Strasbourg site (Fig. 5.1c), nodes are deployed inside a single room in the basement of an office building. In all cases, the distance between adjacent nodes does not exceed 1 m. On each site, we run two types of experiments: an 18 h experiment with 50 nodes, and a multi-day experiment with 5 nodes.

From a hardware/system point of view, the three IoT-lab deployments are equivalent. We run Mercator on the same hardware (the "IoT-lab_M3" node[4]), and use the exact same procedure for all experiments on these three sites.

The 21-node Inria-C SolSystem deployment is done across a single 27 m × 10 m section of an office building floor. About 200 people work in that building, many of them using WiFi extensively. Nodes are not attached to external sensors, each node reports temperature data every 1 s. Unlike all other SolSystem deployments, the Inria-C network is forced to form a star topology (only leaf nodes). This is a requirement for the network to produce the per-frequency statistics we need for Sections 5.5.3 and 5.5.4.

[4] `https://www.iot-lab.info/hardware/m3/`

```
1  {"location": "lille", "tx_length": 100, "start_date":
       "2017-12-19 21:34:57", "stop_date": "2018-01-03 %21:29:25",
       "node_count": 4, "channels": [11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26], "tx_ifdur": 100}
2  datetime, src, dst, channel, mean_rssi, delivery_ratio, tx_count
3  2017-12-19 21:34:57, de-90-73, d7-24-58, 11, -48.99, 1, 100
4  2017-12-19 21:34:57, de-90-73, d9-17-59, 11, -50.39, 1, 100
5  2017-12-19 21:34:57, de-90-73, d7-b1-89, 11, -53.43, 0.9, 90
6  2017-12-19 21:35:08, d7-b1-89, d9-17-59, 11, -39.09, 0.7, 20
7  2017-12-19 21:35:08, d7-b1-89, d7-24-58, 12, -57.41, 1, 100
```

Figure 5.3: The first lines of the `lille_1` dataset.

### 5.3.5 K7: Formating Traces

For our data to be compared and reused in a simulator (see Chapter 6), we needed a common format. We introduce K7, a file format for multi-channel radio connectivity traces. The format is simple: a CSV extension with 7 columns following a given value format.

Each dataset represents one experiment, and consists of a single Comma Separate Values (CSV) file in which the first line contains a JavaScript Object Notation (JSON) formatted set of meta information. Fig. 5.3 shows an example of a dataset file.

Line 1 contains metadata about the experiment, encoded as a JSON string. It contains the location where the experiment was run on (`location`), the length in bytes of the frames sent (`tx_length`), the start date (`start_date`), the end date (`stop_date`), the number of nodes in the experiment (`node_count`), the number of channels used (`channel_count`), and the duration between two frames (`tx_ifdur`).

Line 2 contains the CSV header, the names of the "columns" in the data that follows. The following columns must be present:

- *datetime*. The date time when the measurement was taken, in ISO 8601 format.
- *src*. A unique identifier to identify the node (or interface) that sent the frames. Can be a string or an integer.
- *dst*. A unique identifier to identify the node (or interface) that receives the frames. Can be a string or an integer.
- *channel*. The channel on which the frames where sent on. An integer.
- *mean_rssi*. The mean RSSI value recorded. A signed float.
- *delivery_ratio*. The ratio of frames that where successfully transmitted. A float between 0.0 and 1.0.

- *tx_count*. The number of transmissions on which the measurements where taken on. An integer.

The actual data starts at line 3, with one line per PDR measurements. Line 3 for example reads: node `de-90-73` sent 100 frames to node `d7-24-58` on IEEE802.15 4 channel 11, which received 100% of them, with an average RSSI of -48.99 dBm.

A K7 file must contain the columns listed above, but can also be extended with extra columns. Except for *datetime*, *src* and *dst*, if a value is unknown, the field can be left blank.

A link delivery ratio metric can consider a frame to be successfully transmitted only when the sender receives an acknowledgement message (ACK). We consider that the probability of an ACK loss is small if the radio characteristics are identical. Both communications (DATA and ACK) happen in the same slot, meaning that the same radio frequency is used and the time delta is very small ($<$10ms).

The data format is the same whether it is generated by Mercator or SolSystem. This allows the same tools to be used to analyze both. A SolSystem dataset is more complete as it contains informations about the routing layers (e.g. node's neighbors) and about the devices (e.g. energy spent). K7 contains a subset of a SolSystem dataset, focusing on radio connectivity. Because SolSystem does not provide information on which channel was used to transmit a frame, the channel column is left blank. When reusing the dataset, one can decide whether or not to consider the delivery ratio value as equal in all channels.

## 5.4 Published Datasets

Table 5.1 lists the 11 datasets produced by the deployments listed in Chapter 4 and previously in this Chapter. They contain a total of 2,873,156 PDR measurements, gathered over a cumulative 170,037 mote-hours of operation. To the best of our knowledge, they are, to date, the most comprehensive set of multi-frequency connectivity datasets gathered over a wide variety of environments.

## 5.5 Observations from the Datasets

The datasets presented in Section 5.4 contain a wealth of information. The goal of this section is to contrast/compare the connectivity in testbeds and real-world deployments. We highlight the lessons (we) learned when "moving beyond testbeds", and believe these are interesting to the readership.

Clearly, the points we discuss do not necessarily apply to every testbed, nor do we claim to even know what "realistic" connectivity means (see discus-

| dataset name | # nodes | duration | # PDR measurements | associated figures |
|---|---|---|---|---|
| lille_1 | 5 nodes | 15 days | 367,293 | Figs. 5.6a |
| lille_2 | 50 nodes | 18 h | 274,392 | Figs. 5.4a, 5.5a, 5.8a |
| grenoble_2 | 50 nodes | 18 h | 284,068 | Figs. 5.4b, 5.5b, 5.8b |
| strasbourg_1 | 5 nodes | 3 days | 81,900 | Figs. 5.6b |
| strasbourg_3 | 49 nodes | 21 h | 300,938 | Figs. 5.4c, 5.5c, 5.8c |
| evalab_1 | 22 nodes | 3 days | 9,422 | Figs. 5.8d |
| evalab_2 | 22 nodes | 3 days | 58,895 | Figs. 5.4d |
| smartmarina_1 | 18 nodes | 4 months | 1,122,177 | Figs. 5.4e |
| smartmarina_2 | 19 nodes | 4 months | 183,939 | Figs. 5.8e |
| peach_1 | 19 nodes | 4 months | 166,927 | Figs. 5.8f |
| inria-c | 20 nodes | 30 h | 23,205 | Figs. 5.5d, 5.6c, 5.6d |
| **11 datasets** | **170,037 mote-hours of operation** | | **2,873,156 PDR measurements** | |

Table 5.1: Summary of published datasets.

sion in Section 5.6). That being said, we believe the datasets to be comprehensive enough to extract clear connectivity characteristics in real-world cases that are not per-se present in testbeds. Our main message is that protocol evaluation should be done also in the presence of these different phenomena.

Specifically, this section answers the following questions: *What are the phenomena related to connectivity that are typically seen in real-world deployments? How can these be measured? Are those phenomena present in most testbeds?*

Mercator was created specifically to gather dense datasets; all testbed datasets are hence used in each section below. SolSystem was not created to create these datasets, we hence cannot use all real-world datasets in each analysis. The specificities are:

1. the Peach network does not generate per-frequency information because of outdated firmware,
2. the EvaLab and SmartMarina deployments do generate per-frequency information, but not on a link-by-link basis,
3. the Inria-C dataset is the only one that contains per-link and per-frequency PDR measurements, but is constrained to a star topology.

Based on these constraints, we pick the right datasets to fuel the different discussion points below.

Section 5.5.1 contains preliminary observations about node degree. Sections 5.5.2, 5.5.3 and 5.5.4 then focus on what we argue to be key qualifiers of a deployment environment: *(1)* the amount of external interference, *(2)* the

| | testbed | | | real-world | | |
|---|---|---|---|---|---|---|
| | Lille | Grenoble | Strasbourg | EvaLab | SmartMarina | PEACH |
| Average Node Degree | 49.00 | 38.67 | 48.00 | 11.32 | 5.94 | 9.04 |

Table 5.2: Average degree of a node.

amount of external multi-path fading, and *(3)* the dynamics of the environment, respectively.

### 5.5.1 Node Degree

Average node degree, or the average number of neighbors of the nodes in the network, is typically used to quantify topologies. Table 5.2 shows the node degree in the 6 deployments, using a 0 dBm output power in the testbeds and +8 dBm in real-world deployments. We declare two nodes as being neighbors when the link that interconnects them has a PDR of at least 50%. We borrow this rule from SmartMesh IP[5].

While there is certainly no rule defining what a "realistic" node degree is, typical real-world deployment operators try to cut cost by deploying the smallest possible number of nodes. Analog Devices, for example, recommends that each node has at least 3 neighbors; if given the choice, network operators will not exceed that number. In that case, a node degree around 3 is a lower bound.

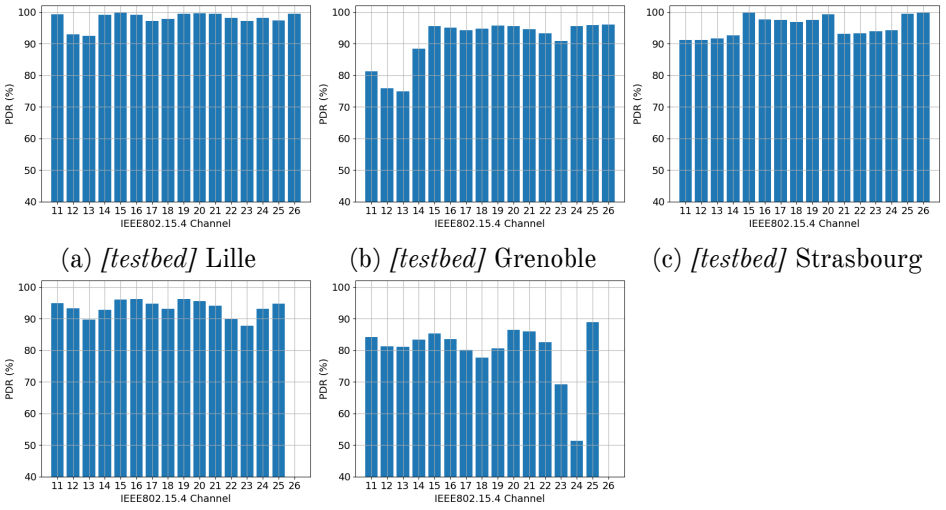
Table 5.2 shows that the testbeds used exhibit a very high node degree, at least 5 times that of the real-world deployments. Testbed operators typically recommend lowering the output power of the nodes to lower the average node degree. Section 5.6.2 argues that this is not a good idea, but that the real solution is to spread the testbed nodes.

> The lesson learned is that testbeds may be too densely deployed (e.g. all nodes in the same room) and that reducing the output power is not a valid workaround.

### 5.5.2 Witnessing External Interference

External interference happens when a different technology – or a different deployment of the same technology – operates within the same radio space. In the types of networks considered in this thesis, the most common case of external interference is IEEE802.11 WiFi interfering with IEEE802.15.4 at 2.4 GHz. WiFi interference causes a portion of the packets sent by the low-power wireless nodes to fail, requiring re-transmissions.

[5] `http://www.linear.com/dust_networks/`




(a) *[testbed]* Lille (b) *[testbed]* Grenoble (c) *[testbed]* Strasbourg

(d) *[real-world]* EvaLab (e) *[real-world]* SmartMarina

Figure 5.4: *[External Interference]* PDR per frequency, averaged over all measurements. IEEE802.15.4 channel 26 (2.480 GHz) is not used by SmartMesh IP, and hence does not appear in the real-world plots.

External interference can be shown by plotting the PDR, averaged over all measurements, grouped by frequency. This is done, for all deployments[6], in Fig. 5.4.

Fig. 5.4 shows some level of external WiFi interference on all deployments, except for IoT-LAB Lille. In a multi-Access-Point WiFi configuration, different APs typically operate on 3 different frequencies, centered around IEEE802.15.4 channels 13, 18 and 23. This is clearly the case in the EvaLab deployment (Fig. 5.4d). It appears from Fig. 5.4b that IEEE802.11 channel 1 (2.412 GHz) is mostly used next to the IoT-LAB Grenoble deployment. In the SmartMarina deployment (Fig. 5.4e), the very high interference on IEEE802.15.4 channels 23-24 is due to a continuously streaming WiFi security camera next to the deployment site, operating on IEEE802.11 channel 11 (2.462 GHz).

The lesson learned is that external interference from WiFi is typically present in real-world deployments. It is also most often present in testbeds, as those are typically deployed in office buildings.

[6] The appropriate HRs data was not gathered on the SolSystem Peach deployment; we are hence unable to plot the figure for that deployment.

### 5.5.3 Witnessing Instantaneous Multi-Path Fading

Multi-path fading is both less intuitive and far more destructive than external interference. It is entirely caused by the environment around nodes that communicate. When node `A` sends a frame to node `B`, what `B` receives is the signal that has traveled over the line-of-sight path between `A` and `B`, but also the "echoes" that have bounced off of nearby objects. Depending on the relative position of nodes `A` and `B` and the objects around, these different components can destructively interfere. The result is that, even though `A` and `B` are close, and that `A` transmits with a high output power, `B` does not receive any of its frames. This "self-interference" pattern depends on the frequency used. What typically happens is that node `A` can send frames to node `B` on most of the available frequencies, except on a handful of frequencies on which communication is impossible. The impact of multi-path fading is higher when the deployment area is cluttered by highly reflective (e.g. metallic) objects.

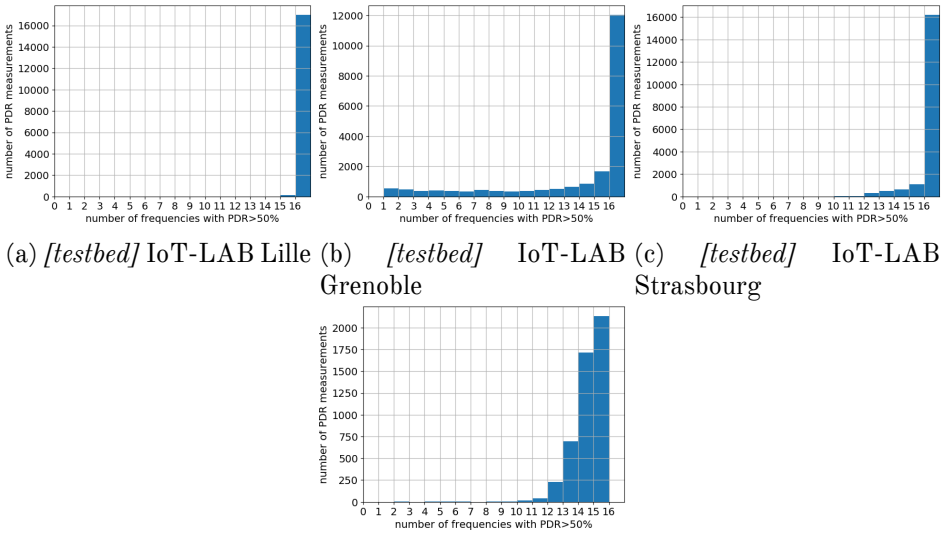
What we are looking for in the datasets is hence how many frequencies are usable (PDR>50%), for each link. If all frequencies are usable, there is no multi-path in the environment. Fig. 5.5 plots, for each PDR measurement, how many frequencies have a PDR higher than 50%.

In the IoT-LAB Lille case (Fig. 5.5a), almost all PDR measurements show that all frequencies are usable: there is very little multi-path fading in that environment. This is expected, as the deployment is done in one large uncluttered room (see Fig. 5.1a). In contrast, multi-path fading is very present in the IoT-LAB Grenoble site (Fig. 5.5b). This is expected, as the deployment is done in a tight space between the dropped ceiling and the roof, a space cluttered with metallic structure and wiring (see Fig. 5.1b). Multi-path fading is also very present in the Inria-C deployment (Fig. 5.5b). This deployment spans multiple rooms, with 20 m long links crossing several walls and rooms filled with white boards, chairs, tables, ventilation piping, etc., all opportunities for multi-path fading to occur.

> Multi-path fading takes place in varying degrees in virtually all deployments. It is in particular present in an environment cluttered with highly reflective (e.g. metallic) objects, or simply when links are long (over 10 m). It causes the PDR of a link to vary significantly with frequency, and it is essential to test networks in testbeds in which there is a lot of multi-path. The lesson learned is that it is essential to deploy a testbed across a large area, e.g. across an entire floor rather than in a single room.

### 5.5.4 Witnessing Dynamics in the Environment

In virtually any real-world deployment, the environment changes over time: people move across buildings, WiFi traffic continuously changes, machines are




(a) *[testbed]* IoT-LAB Lille (b) *[testbed]* IoT-LAB Grenoble (c) *[testbed]* IoT-LAB Strasbourg

(d) *[real-world]* SolSystem Inria-C

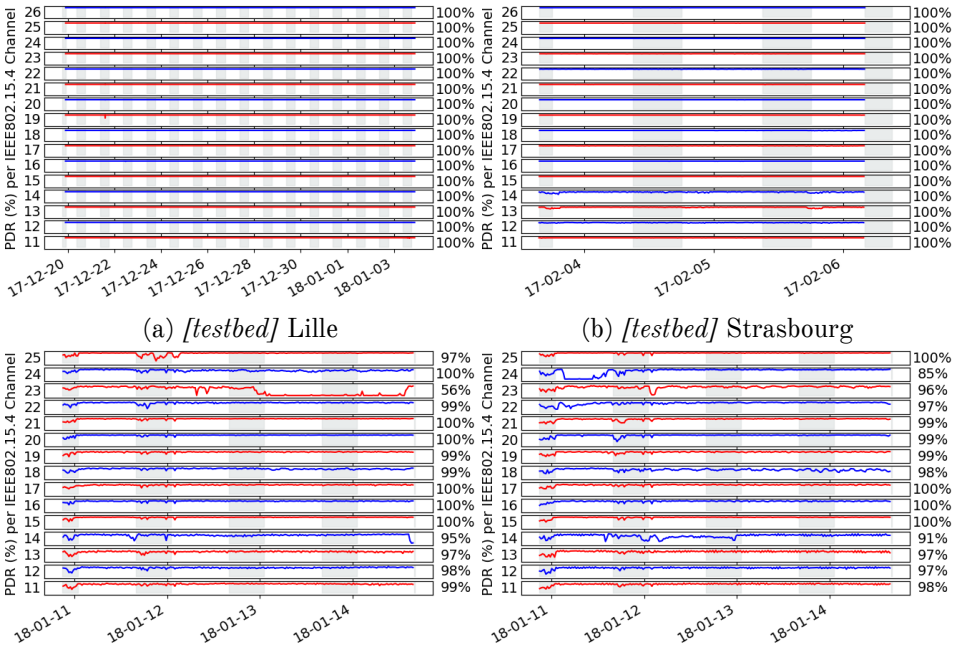
Figure 5.5: *[Instantaneous Multi-Path Fading]* Measurements with number of frequencies with PDR>50%.

switched on and off, doors are opened and closed, forklifts zip around factory floors, etc. This means that the level of both external interference and multi-path fading changes over time. From a connectivity point of view, this means that the PDR of each link varies over time, and across all frequencies.

Fig. 5.6 shows how the PDR of particular links varies over time, on each IEEE802.15.4 frequency. The gray zones highlight daily business hours. While we had to choose specific links for each deployment, we make sure they are representative of the other links.

In the Inria-C deployment, Figs. 5.6c and 5.6d show the PDR variation over time for the link from nodes $TX1$ and $TX2$ sending to node $RX$, respectively. Nodes $TX1$ and $TX2$ are both placed 27 m away from $RX$. Even though $TX1$ and $TX2$ are only separated by 50 cm, the per-frequency PDR variations on their links to node $RX$ evolve in very different manners, which is expected.

Figs. 5.6a and 5.6b show the variation of PDR on a particular link in the IoT-LAB Lille and IoT-LAB Strasbourg deployment, respectively. Even over many days, there are no significant changes in PDR. This has severe consequences, as a networking solution validated on a testbed like this might fail in the real world, in which the environment (and the PDR) changes frequently.




(a) *[testbed]* Lille

(b) *[testbed]* Strasbourg

(c) *[real-world]* Inria-C TX1 → RX

(d) *[real-world]* Inria-C TX2 → RX

Figure 5.6: *[dynamics in the environment]* PDR evolving over time for specific links.

In virtually all real-world deployments, the environment in which nodes are deployed changes, resulting in dynamics in the connectivity between nodes, on each frequency. Testbeds often do not capture these effects, as nodes may be deployed in basements. This has a severe impact on the validity of evaluations in these testbeds, and solutions working perfectly on them might not work at all in the real world. The lesson learned is that the evaluation of a networking solution on a testbed without dynamics has very limited validity.

## 5.6 Discussion

Section 5.4 has generated 11 dense connectivity datasets gathered over both testbeds and real-world deployments. Section 5.5 analyzes the node degree, level of external interference and multi-path fading, and their variation over time, and contrasts/compares testbeds and real-world deployments. The goal of this section is to discuss what changes when going from testbeds to real-world deployments. In particular, we discuss some of the steps one needs to take to ensure a solution is properly tested in a testbed so it succeeds in

real-world deployments.

We start by discussing what "realistic" connectivity looks like, and propose a network evaluation checklist in Section 5.6.1. We quickly focus on the common misconception that lowering the output power is a good idea (Section 5.6.2). In Section 5.6.3, we then introduce the notion of a waterfall plot, and how to use it to visualize the connectivity in a deployment. Finally, Section 5.6.4 presents directions for future work.

### 5.6.1 What is Realistic?

We do not claim to know what "realistic" connectivity looks like. Every deployment is different, and a dense deployment in a small basement room is as realistic as a deployment on an entire factor floor. It all depends on the application. We *do not* argue in favor or against particular testbeds.

Rather, we list the 3 phenomena that are most common in real-world deployments, and which have a deep impact on connectivity: external interference, multi-path fading, dynamics in the environment. Any deployment exhibits a combination of these three phenomena. **When evaluating a networking solution, it is hence essential to do so in environment(s) which exhibit all three. Without this, you run the risk of having your solution fail during a real-world deployment.**

Before evaluating a solution in a testbed, we recommend you go through the following 5-point checklist:

1. Gather connectivity traces that are dense in time, frequency and space, by using Mercator, SolSystem, or an equivalent tool.
2. Compute the average node degree (as in Section 5.5.1), and ensure that you are testing your solution also on very sparse deployments (down to a degree of 3).
3. Plot the average PDR for each frequency (as in Section 5.5.2), and ensure that you see variations across different frequencies, indicating the presence of external interference.
4. Plot a histogram of the number of frequencies with PDR>50% (as in Section 5.5.3), and ensure that a significant portion of the links in your deployment have one or more "bad" frequencies, indicating the presence of multi-path fading.
5. Plot, for each link, the evolution of its PDR over time, for each frequency (as in Section 5.5.4), and ensure that a significant portion of the links see the PDR switch from 0% to 100% on multiple frequencies, indicating the presence of dynamics in the environment.

It is our experience that a solution evaluated on a testbed in which the check-list above passes performs well in real-world deployments.

### 5.6.2 A Word about Output Power Tuning

Some testbeds are often too densely deployed, and to limit the node degree (number of neighbors) and increase the network radius (number of hops), testbed operators often reduce the output power of the radios (e.g. −55 dBm). This is not a good idea. The reason is that this also limits the amount of multi-path fading, as the echoes that reach the receiver antenna are so weak that self-interference is not happening. The result is a deployment that looks more like free space.

Instead, we recommend installing MAC address filtering on the software of the nodes so they artificially drop packets from neighbors not in the list. This is a way to force a topology while maintaining the same level of multi-path fading and dynamics.

### 5.6.3 Waterfall Plot

Each PDR measurement in the datasets also contains the average RSSI over the 100 frames received in that burst. Plotting a scatterplot of PDR as a function of RSSI reveals a large number of insights about the connectivity in the network. Because of its shape, we call this a "waterfall plot". In the absence of external interference and multi-path fading, the waterfall plot is at PDR≈100% above sensitivity, at PDR≈0% 10-15 dB below the radio chip's sensitivity, and with a almost linear ramp between the two.

You can apply the tools detailed in this section both to your testbed (to verify it is "realistic"), and to your real-world deployment (to quantify its connectivity).

We assume you have generated a waterfall plot from the connectivity dataset gathered in your deployment. Fig. 5.7 shows such a waterfall plot. Each cross represents a PDR measurement; the mean value with standard deviation is also depicted. Fig. 5.7 contains annotations on how to "read" it:

① Make sure the left-hand side of the waterfall plot is complete, i.e. it reaches 0%. Not having this left-hand side indicates that your nodes are very close to one another. On a testbed, this means you are not testing your solution close to sensitivity.

② Any discontinuity in the plot indicates that your deployment contains either very good links, or bad links, but no in-between. This is typically the case for networks in which nodes are deployed in clusters.

③ A waterfall plot shifted to the right indicates the presence of external interference and multi-path fading.

④ A "dip" in the waterfall plot indicates strong interference on specific links.

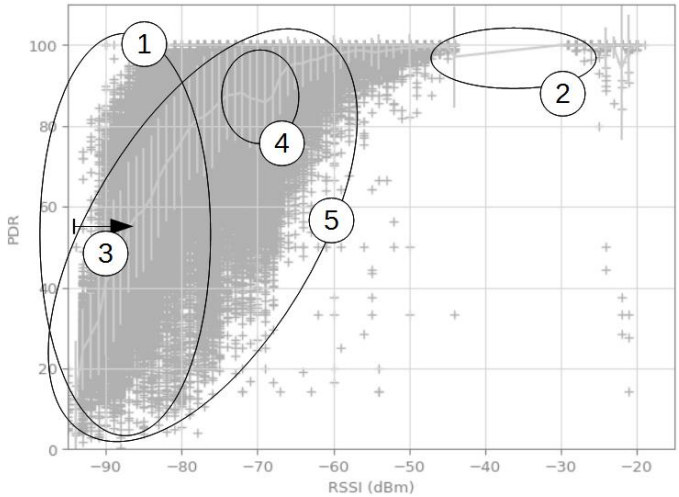



Figure 5.7: Five elements to look at when assessing the connectivity in a deployment by "reading" its waterfall plot (detailed in Section 5.6.3).

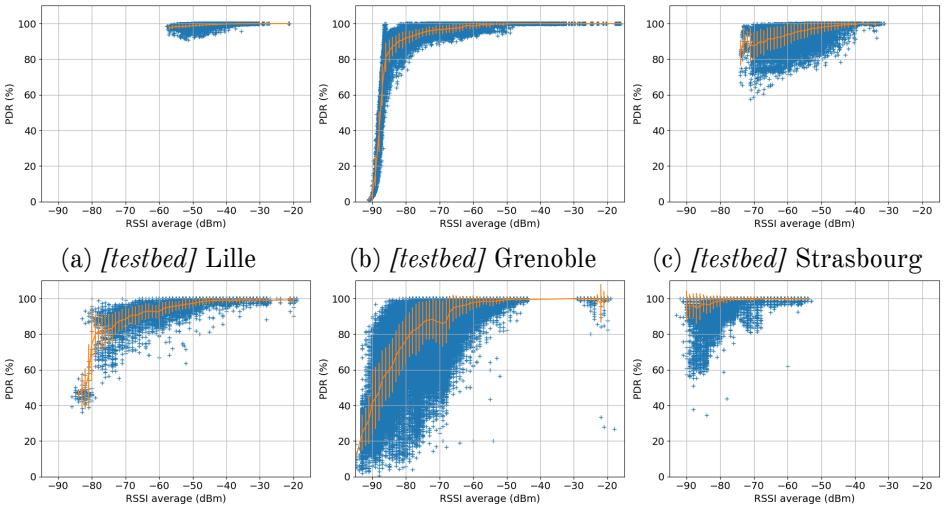
⑤ The spread of PDR measurements around the mean value indicates dynamics in the environment.

Given these rules, just looking at a waterfall plot allows one to determine how close together nodes are deployed, and whether external interference, multi-path fading and dynamics are present.

We show the waterfall plots for all deployments in Fig. 5.8. The rules above allow us to get good insights into the connectivity in the deployments (circled number refer to the rules above). The IoT-LAB Lille and Strasbourg testbeds (Figs. 5.8a and 5.8c) suffer from the fact that nodes are deployed too close to one another ①. Nodes are deployed in clusters in SmartMarina, as shown by the discontinuity in the plot ②. The fact that the EvaLab and SmartMarina waterfall plot are shifted right compared to Peach indicates external interference in the former two, very little in the latter ③. A WiFi camera interferes with a small number of links in SmartMarina; this can be seen by the "dip" in the plot ④. Nodes in the IoT-LAB Grenoble testbed are deployed far enough apart from each other, but lacks dynamics in the environment ⑤.

### 5.6.4 Directions for Future Work

The format of the dataset highlighted in Section 5.4 is generic enough to include additional datasets. There would be great value in creating a standardized connectivity evaluation kit and deploy it in various environments for several weeks, in order to generate a comprehensive set of connectivity datasets.




(a) *[testbed]* Lille (b) *[testbed]* Grenoble (c) *[testbed]* Strasbourg

(d) *[real-world]* EvaLab (e) *[real-world]* SmartMarina (f) *[real-world]* PEACH

Figure 5.8: Waterfall plots for the different deployments.

Simulation platforms could be modified to "replay" these connectivity datasets, rather than relying on propagation models at the physical layer. The benefits would be that *(1)* this would increase the realism and confidence in the simulation results, and *(2)* the same simulation could be run against a number of datasets, which would serve as connectivity scenarios.

There would be great value in defining a set of metrics to quantify how much external interference, multi-path fading and dynamics there is in a network. Networking solution could be benchmarked against several deployments, covering a range of metrics.

Similarly, it would be interesting to evaluate how much the type of connectivity impacts the performance of networking solution, such as those proposed by the academic community.

## 5.7 Summary

In this chapter, we compare results obtained from real-world networks with ones obtained by running radio connectivity experiments on testbeds. Having dense radio connectivity information allows us to have a closer look of what happens at the MAC layer. We identify key metrics and behaviors that we must take into account when developing a protocol for the IIoT and propose a methodology to ensure that a protocol that performs well on a testbed also does so when moving beyond testbeds. More specifically, we look at External Interferences, Multipath Fading, and their variations over time.

Now that we have more insight of what happens at the MAC level, we want

to study TSCH in more details and find its limits and trade-offs. In Chapter 6, we present and use the 6TiSCH Simulator to study the capacity of TSCH in terms of end-to-end latency, reliability and energy consumption.

# Chapter 6

# TSCH Limits and Trade-offs

**In this chapter, we analyze the limits and trade-offs of TSCH in term of end-to-end latency and network lifetime. We compare the performance results obtained by simulation with that of real deployments, and propose a tool to estimate the performance of 6TiSCH networks.**

Based on what we learned from our previous work, we study the limits and trade-offs of TSCH in terms of end-to-end (E2E) latency, E2E reliability, and energy consumption (battery lifetime). Section 6.1 defines those concepts and explains their theoretical limits. Section 6.2 presents the 6TiSCH simulator that we used to compare real-world network performances against the theoretical limits. Finally, we introduce the 6TiSCH Performance Estimator, a tool we developed to predict the performance of a 6TiSCH network.

## 6.1 Theoretical Limits

In this section, we assess the performance limits of TSCH under certain assumptions. We consider the E2E reliability as a fixed requirement, and tune the TSCH schedule to explore the trade-off between latency and energy consumption. We start by defining the set of assumptions we make, then define what we mean by reliability, latency and energy consumption, before presenting theoretical results under various hypothesis.

I want to answer the following questions:

- Given a 99.999% minimum E2E reliability, what is the minimal latency a network can achieve?
- By how much does the network lifetime increase when relaxing the latency constraint?

### 6.1.1 Assumptions

**We assume that each device has a unique radio interface.** That is, a device can only be assigned to one cell at a time. A node cannot receive and transmit at the same time and cannot receive from multiple other nodes at the same time. This is a fair assumption, given the devices on the market today.

**We only consider convergecast traffic, that is, all the nodes in the network send their traffic to the gateway.** This assumption reduces the application scope to monitoring and alert detection application, and does not encompass applications that require actuation (triggered by the manager or other nodes). While we strongly agree that other scenarios should be studied too, we consider it as a second step and present our vision towards it in Chapter 7.

**We consider monitoring applications (application that generate periodic traffic) and time-critical alerting applications as identical from a scheduling point of view**. The same amount of resources (cells) have to be allocated for both types of applications. While in time-critical alerting applications those allocated cells will are *not* used most of the time, they still need to be present as reactive allocation would induce large delays.

**As this is the default 6TiSCH behavior, we only consider routing topologies where each node has only one parent.** While we understand this is a minimal configuration to prove the effectiveness of 6TiSCH, we strongly believe that topologies with multiple parents should be adopted in IIoT as their offer more stability (e.g. fallback links) and routes diversity (e.g. load balancing).

### 6.1.2 Key Performance Indicators

#### E2E Upstream Reliability

For low-power wireless technologies to be adopted by the industry, they need to provide wire-like end-to-end (E2E) reliability. The end-to-end reliability can be expressed as the amount of messages that reach their destination, given a total number of messages generated in the network. As an example, SmartMesh IP is a TSCH-based commercial wireless sensor technology that yields over 99.999% E2E reliability. E2E Reliability is the first parameter we look at.

If a packet is too long to fit in an IEEE802.15.4 frame (>127 B, including headers), it can be fragmented and sent over user multiple frames. The packet can only be considered as received if all of its fragments are received. **We define the "E2E Upstream reliability" as the packet reliability between a node and the network root.**

#### E2E Upstream Latency

The E2E Upstream latency is a key parameter in time-critical applications. It is the time delta between the generation of an application packet and its

arrival at its final destination.

**We define the "E2E Upstream latency" as the time a packet takes from the moment it was generated in a node to the moment it was received by the manager.** The latency can be expressed in seconds, for easy understanding, or in slots, for easy comparison (as the duration of a timeslot can differ between implementations). We consider that the time between the packet generation by application layer and its reception as frames in the MAC layer as negligible.

Latency cannot be bounded. In other words, we cannot claim that all packets (or frames) are going to be delivered within a given time. This happens because the probability of an infinite number of consecutive retransmissions is not null. Instead, we can only provide latency guarantees in "most cases". In my work, I use the latency distribution percentile as a mean to quantify "most cases". For instance, I claim that, given a certain set of conditions, the maximum E2E upstream latency is going to be X seconds for X% of the packets.

#### Network Lifetime

It is commonly agreed that the lifetime of a node is the time before it runs out of battery. But looking at the lifetime of a network, three definitions can come to mind: the time before every node runs out of battery, the average node lifetime, or the time before the first node runs out of battery. In industrial applications, the last definition is often the most appropriate. Indeed, we are interested in applications where each sensing point is critical. **We define the network lifetime as the time until first node runs out of battery**.

### 6.1.3 Objectives

#### Collision-Free Schedule

The radio connectivity traces we collected do not contain information about how the radio connectivity behaves when there are multiple transmissions on the same channel at the same time. Thus, rather than using a theoretical model which would estimate a link quality when multiple transmissions occur at the same time and on the same channel, we prefer to make sure this never happens by building a schedule that guarantees such a property. **We want to build a collision-free schedule.**

#### Minimizing the Maximum Latency

Our first step is to provide latency bounds so that network operators can quantify the worst case latency they can expect, that is the maximum E2E latency. We also want latency to be as low as possible to know if a low-power wireless network can tackle time-critical applications. **We want to build a schedule that minimizes the maximum E2E upstream latency.**

### 6.1.4 A Canonical Case

We start with a canonical case to have a better understanding of the challenge. We consider a network with perfect links and that follows the assumptions presented above.

We define a node's *Load* as the number of cells it requires to transmit the frames it generates locally (later denoted as $Gen(n)$) as well as the number of cells it requires to forward (i.e. receive and transmit) the frames generated by its descendants (later denoted as $Fwd(n)$).

#### Flows

We define a *flow* as the series of frames that travel from a source to a destination. In the following, we allocate cells along a path from a source to a destination, and assume that those cells are available for any incoming frame generated by the source. This is true only if flow-isolation or flow-priority is assigned to a cell. Indeed, any node along the path from a source to a destination could use a cell allocated for the source's frames, and make that cell unavailable when a source's frame wants to use it. As an example, in Fig. 6.2, let's assume nodes `C` and `D` generate frames $t_C$ and $t_D$, both addressed to node `A`. If node `C` decides to use its second slot for frame $t_C$, then frame $t_D$ will have to use the next available slot, and its latency will increase.

There are two ways to avoid such situation:

- **flow-isolation**: Cells among a path are reserved for exactly one flow.
- **flow-prioritization**: A flow can be used by any frame, but gives priority to the frames belonging to that flow.

In our canonical case, both ways ensure that all cells are available for a flow so that (6.2) is true. Flow-prioritization would however lead to lower average latency as frames would have more resource (i.e. cells) available.

#### Schedule Lower Bound

In their publication, Khoufi *et al.* calculate the minimum number of slots required for a convergecast application of a network using TSCH with perfect links (no retransmissions) **khoufi17scheduling** They prove that the minimum number of slots assigned to nodes is lower bounded by $max(S_n, S_t)$, with:

$$max(S_n, S_t)S_n = \sum Gen(i)S_t = Gen(c_1) + 2\sum Gen(v) \tag{6.1}$$

$S_n$ indicates the number of cells required by the gateway to receive all the frames generated by the network without retransmission. $S_t$ indicates the number of cells required by the first-hop node with the highest *Load* ($c_1$). We simplified $S_n$ and $S_t$ as we only consider nodes with one network interface.

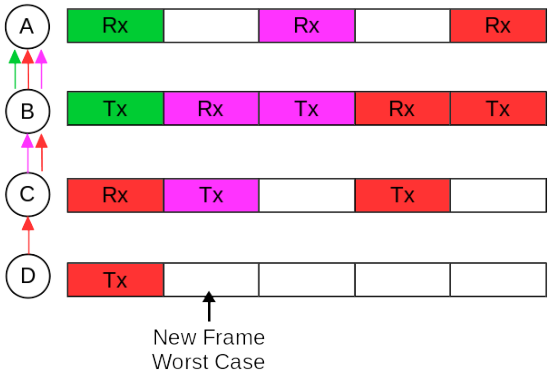



Figure 6.1: Valid Cascading Schedule. New frame worst case scenario. Max E2E US latency is 8 slots.

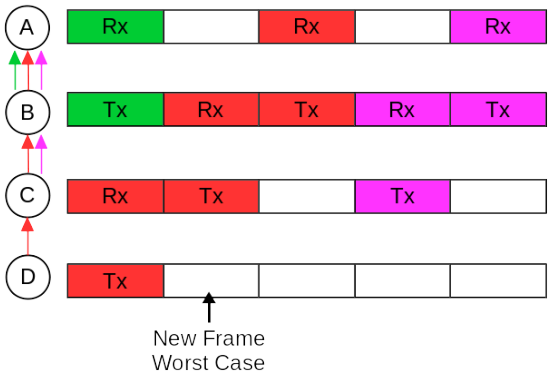



Figure 6.2: Optimal Cascading Schedule. New frame worst case scenario. Max E2E US latency is 7 slots.

### Latency Upper Bound

Fig. 6.1 depicts a topology and its corresponding schedule that fits in the slotframe size optimal bounds. That is, it delivers all the packets within a 5-slot slotframe (i.e. $S_t$=5).

A schedule that fits the optimal slotframe size is not necessarily a schedule that is optimal in terms of E2E US latency. As an example, Fig. 6.2 shows a schedule that has a smaller maximum E2E US Latency than the schedule presented in Fig. 6.1.

The minimal latency is achieved when the series (i.e. a cascade) of pairs of cells (i.e. links) between the source and the destination are consecutive in time. Without retransmissions, this corresponds to the number of links that separate a source node to the destination. In convergecast traffic, this is the depth of a node. As we consider a frame lifetime to start when the frame is

given to the MAC layer, the worst case occurs when the frame is given right after the cascade starts. As we assume one cascade per slotframe, the time before the cascade restarts is equal to the size of a slotframe -1. The maximum E2E upstream latency is thus defined as:

$$Min((SlotframeSize - 1) + Depth(n)) \tag{6.2}$$

For each node, where $n$ is the node and $Depth(n)$ the depth of node $n$.

**Lifetime Bounds**

The energy spent by a mote can be directly derived by the number and type of cells in its schedule **vilajosana14realistic** We first consider that a cell requires the same electric charge whether it is used or not. In the schedule presented in Fig. 6.2, `B` is the node with the maximum number of cells and its average energy consumption can be expressed as:

$$(3 \times TxCharge + 2 \times RxCharge)/SFLength \tag{6.3}$$

Knowing this, we can increase the lifetime of a mote by increasing the size of its slotframe with Idle cells. This reduces the average energy consumption but also increases latency. The maximum average consumption of a node can be expressed as:

$$\frac{MinSFLength \times ActiveAvgCharge + SlotIncrease \times IdleCharge}{MinSFLength + SlotIncrease} \tag{6.4}$$

Where $ActiveAvgCharge$ is the average charge spent during the minimal slotframe, and $IdleCharge$ is the charge spent during one Idle slotframe (i.e. a slotframe with not transmissions or receptions scheduled). Fig. 6.3 depicts the lifetime vs. latency trade-off.

For instance, let's assume an $IdleCharge$ of 1 $\mu A$, and an $ActiveAvgCharge$ of 300 $\mu A$ with a 510 slots MinSFLength. To increase the lifetime by one year, 443 idle slots would need to be added. Taking a 7.25 ms slot, this would increase the latency by 3.21 s.

Using those equations can give a rough idea of the estimated lifetime of a mote before any communication starts, however, those equations can be refined at run time. We considered that cells would use the same electric charge whether they are used or not, and this is false in most implementations. **RxCells**. When in an RxCell, a node listen for incoming traffic. If no traffic arrives within the first ms (configuration dependent), the node knows that no traffic will arrive in the rest of that cell and can thus turn its radio off. By doing so, a node consumes a few mC per cell (e.g. 4 mC for the LTC5800-IPM). **TxCell**. When in a TxCell, if a node has not frame to send, it will not start its radio. The cell can thus be considered as an IdleCell (e.g. 52 mC saving

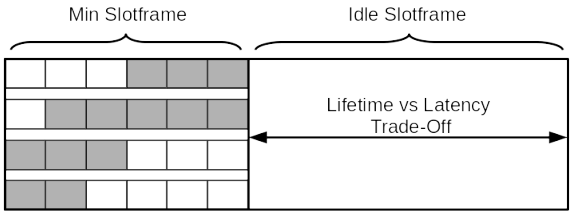



Figure 6.3: We can tune the lifetime vs. latency trade-off by changing the slotframe size.

per cell for the LTC5800-IPM). **ACK**. When in a Tx or RxCell, if no ACK frame is required, both communicating nodes can turn there radio off after the DATA frame exchange (e.g. 4 mC saving per cell for the LTC5800-IPM).

The lifetime of a mote can thus be expressed as a range between the best case scenario (when no cells are used) and the worst case (when all cells are used with ACKs).

We defined the network lifetime as the time before the first node runs out of battery. The network lifetime can thus be expressed as:

$$Min(lifetime(n)), \forall n \in Nodes \tag{6.5}$$

Note that those estimations only take into account application packets. A finer energy consumption estimation should also account for network control traffic.

### 6.1.5 With Retransmissions

Assuming perfect links, Eq. 6.9 allows us to calculate the minimal slotframe size, Eq. 6.2 the maximal E2E US latency, and Eq.6.5 the network lifetime. We now try to find corresponding equations considering links where a certain number of retransmissions must be made before one succeeds. Our goal is to build a schedule that allocates enough cells so that a frame, even when retransmitted multiple times, gets to its final destination within one slotframe. The challenge is to find the right number of retransmissions a link (or an E2E path) should be able to handle.

Pöttner *et al.* **pottner14constructing** introduce $B_{max}$, the maximum count of consecutive transmissions needed for a frame to be transmitted on a link. This metric is obtained empirically for each link based on previous transmissions and can be seen as an *observed* worst case. Then, they build a schedule that allocates $B_{max}$ cells for every frame than a link is supposed to carry. The problem with such a method is that it may require a long delay (authors shows that it could take more than 140 hours) before a representative $B_{max}$ is known for all the links.

We use a different approach and derive the maximum number of transmissions required before success from the Packet Delivery Ratio (PDR). Such a metric does not provide information about when the failure or success happens; so for every transmissions, there is a probability of $P = PDR\%$ of success and $1 - P$ of failure. The probability of success after $k$ consecutive transmissions attempts on the same link is equal to the probability of $k$ consecutive failures followed by the probability of success, that is, $(1 - P)^k \cdot P$. We want to find $k$, the number of consecutive failures before a successful transmission, so that the probability of success after $k$ transmissions must be over the E2E reliability we want the network to achieve. Said differently (this leads to a simpler equation), we want to find the probability of non reception $(1 - R)$ after $k$ consecutive transmissions. We express this problem in (6.6), where $P_{ab}$ is the PDR of the link from node `A` to node `B`, and $R$ is the expected E2E US reliability.

$$(1 - P_{ab}) \leq 1 - R \tag{6.6}$$

This leads to (6.7). For instance, to achieve an E2E reliability $R$ of $99.999\%$ with a link of PDR=$70\%$ ($P_{ba} = 0.7$), a node `B` at one hop of the root `A` would need to transmit $K \geq 9.56$, that is, 10 times.

$$k \geq \frac{log(1 - R)}{log(1 - P_{ab})} \tag{6.7}$$

Eq. 6.7 allows us to compute the number of transmissions required before a successful transmission for one link (same result is shown in **dobslaw16end**). As the number of retransmissions is an integer, we usually round the result up.

As we want to guarantee an E2E US reliability, we need to extend this equation to work with multiple consecutive links. Applying (6.7) to each link of the path from source to gateway does not work, as the probability depends on the delivery of the previous link. This would lead to $R' = R^{\#hops}$ and because $R < 1$; the resulting reliability would be inferior to the expected reliability ($R' < R$).

We need to find which reliability ($R_i$) each link must provide in order to fulfill the E2E reliability. There are many combinations of multiplications that could result in $R$. In their article **gaillard16enabling** Gaillard *et al.* propose a greedy algorithm that optimizes the distribution of links network-wide. In our case, we consider the optimal solution as the one that minimizes the total number of transmissions.

$$Min(\sum_{i \in links_f} k_i) \tag{6.8}$$

**Schedule Lower Bound**

Unlike when building schedules with perfect links (see Section 6.1.4), the most loaded node is not necessarily the root or one of the first hops. There could be nodes with a high Load because of retransmissions, and not because they have many descendants.

Empirically, we find that unless the number of concurrent transmissions is superior to the number of available channels, Eq. 6.9 allows us to calculate the minimal slotframe size. $k_{ab}$ corresponds to the number of transmissions required for node `A` to transmit a frame to node `B`.

$$\begin{aligned} &Max( \quad TxCells \quad + \quad RxCells \quad ) \\ &Max((Gen(n) + Fwd(n)) \times k_{np} + \sum_{c \in preds} (Gen(c) \times k_{cn}))) \end{aligned} \tag{6.9}$$

**Latency Upper Bound**

Like in Section 6.1.4, the worst frame acquisition time is right after the cascade starts. If this happens, the first opportunity to transmit the frame arrives $Slotframe Size - 1$ slots later. Then, in the worst conditions, the frame needs the maximum number of retransmissions per hop to reach the destination. We call $Cascade Size$ the time between the first and the last slot of the cascade. The minimal maximum E2E upstream latency is defined by (6.10).

$$Min((Slotframe Size - 1) + Cascade Size) \tag{6.10}$$

**Lifetime Bounds**

The lifetime bounds can be estimated the same way than the ones presented in Section 6.1.4.

### 6.1.6 Conclusion

In this section, we show that, theoretically, we can provide estimated performances in term of E2E upstream latency and network lifetime. We use the equations provided in this section to estimate the limits a TSCH-based network can achieve, given a routing topology and link quality informations. In the following sections, we test those equations against real-world network performance using the 6TiSCH simulator, a simulator dedicated to 6TiSCH networks.

## 6.2 Simulating the IIoT

Before a company adopts (and invests in) 6TiSCH, it needs to know whether the technology works for its applications. This means answering very down-to-earth questions about the performance of the network: *What is the latency distribution 6TiSCH would offer in this particular industrial deployment? If I use 6TiSCH in my smart agriculture application, how often will I have to replace batteries? Would 6TiSCH offer over 99.999% end-to-end reliability in this particular smart parking deployment?* What is needed to answer these questions is a way of **benchmarking** the solution standardized today. That is, we need a tool which can turn a particular deployment and traffic pattern in key networking performance indicators such as per-node battery lifetime, end-to-end reliability, and latency distribution.

Of course, it would be possible to answer these questions through a comprehensive real-world experimentation campaign. And while extensive experimentation is happening (and will be happening more and more), we argue that it lacks the flexibility of doing quick "what-if" analysis. For that, simulation offers several advantages. First, it allows us to quickly estimate the performance for a given scenario and a set of parameters. Second, it allows us to tune the parameters in a flexible manner until the desired performance is reached, and map these values to the real-world deployment. That being said, simulation is necessarily an imperfect model of physical reality, in particular the radio propagation/connectivity between nodes. And while simulation results must be taken with a grain of salt, they are a necessary first step before any deployment (discussed further in Section 5.2).

To conduct our work, we combine the large datasets we obtain from previous work together with the knowledge we gain from comparing them.

We compare real-world network results with optimized network results obtained using Trace-Based Simulation in Section 6.3. We present the 6TiSCH Performance Estimator in Section 6.4, a tool to predict 6TiSCH network performance.

### 6.2.1 Related Work

Network simulation is a common approach used in the design, implementation and performance evaluation of different algorithms and protocols when there is a need for assessing the behavior of a network given a set of models and constraints. In the low-power wireless case, discrete event-driven network simulators are widely used since *(1)* they can be used as a flexible and inexpensive tool for testing the network without the need of having to deploy an actual physical network, and *(2)* they can be more accurate and realistic than mathematical models where usually simplifications and abstractions are assumed.

Although real-world deployments are the most reliable approach for eval-

uating network performance, it involves important practical difficulties (logistically and economically), not only for deployment and testing but also for modifying and debugging the network. This is especially relevant in large-scale deployments. Unlike real-world deployments, network simulators allow us to flexibly switch and exchange the different protocols and layers in the stack and quickly evaluate network performance for a number of configurations and scenarios.

Mathematical models and problem formulations, although very relevant for describing and predicting behaviors, sometimes lack the accuracy present not only in the standards and RFCs but also aspects inherent to real-world deployments. In this sense, network simulators can provide the required standard-compliant accuracy in a flexible manner. Additionally, complex models become impossible to solve in acceptable time when the network size increases. This makes network simulators a crucial research tool for bridging the mathematical models with real-world deployments.

The open-source network simulators NS-2 **ns2** and JSim **tyan02design** have been traditionally used for simulating low-power wireless networks. Currently, *NS-3* **ns3** *OMNet++* **omnetplusplus** and *TOSSIM* **levis03tossim** are arguably the most widely used. However, none of these simulators fulfill the goals for which the 6TiSCH Simulator was designed: compliance with the standard, scalability and simplicity.

- *NS-3*, an advanced and more modular version of NS-2, is probably the most widely used network simulator today. It has a number of modules for different technologies and protocols, and the support of a big community of users. Although NS-3 is a very powerful simulator, its complexity and generic purpose involve a significant learning curve and non-significant programming time for a non-expert user. No 6TiSCH implementation currently exists.

- *OMNet++* is also a very extensively used simulator for simulating low-power wireless networks. Its INET framework implements most of the more common network protocols. A very realistic PHY layer is available in Castalia **castalia** an OMNet++-based simulator which implements accurate wireless channels and radio access models. As in NS-3, no 6TiSCH implementation is available.

- *TOSSIM* is a simulator for TinyOS **levis05tinyos** Although TinyOS has been one of the first low-power wireless implementations, support has officially ended in 2013.

Other emulation options are available, including *Cooja* **dunkels02contiki** (the Contiki emulator)[1] and *OpenSim* **watteyne12openwsn** (the OpenWSN

[1] A less detailed, but fully-simulated version is also available, but is still significantly time consuming.

| Simulator | Learning Curve | Scalability | 6TiSCH implementation | Standard-Compliant |
|---|---|---|---|---|
| ns-3 | High | Medium | None | N/A |
| OMNet++ | High | Medium | None | N/A |
| TOSSIM | Medium | High | None | N/A |
| Cooja (emulator) | High | Low | Yes (Partial) | Partially |
| OpenSim (emulator) | High | Low | Yes | Yes (byte-accurate) |
| 6TiSCH Simulator | Low | High | Yes | Yes (behavioral) |

Table 6.1: Comparison between the 6TiSCH Simulator and the different network simulator alternatives.

emulator). These platforms are very powerful for emulating the behavior of a 6TiSCH network without needing the actual hardware, since they run the same binary as goes on a low-power wireless device. Yet, the emulation approach has two main drawbacks. First, unlike discrete event-driven simulators, their scalability is poor, since the real-time requirements of the instances limits the size of the network up to few tens of nodes. Second, due to its bit-level accuracy, new implementations in routing, scheduling functions or traffic management require significantly more effort than in discrete event-driven simulators where some functions can be abstracted.

A comparison between the different existing alternatives is shown in Table 6.1. It evidences why the 6TiSCH Simulator has been developed. First, it has a 6TiSCH implementation that follows the requirements specified in the 6TiSCH RFCs and drafts (e.g. Cooja does not implement MSF). Second, it scales up easily up to several hundreds of nodes. Finally, it has low complexity, enabling to quickly implement and test different scheduling functions, RPL objective functions and traffic management strategies. These points are detailed in Section 6.2.2. In that sense, the 6TiSCH simulator is not really comparable to NS-3 or OMNet++, since they are generic purpose simulators which contain a high number of libraries supporting different protocols and models. The 6TiSCH simulator is a highly specialized protocol specific tool for fast prototyping.

### 6.2.2 6TiSCH Simulator

The 6TiSCH Simulator is a discrete-event simulator written in Python. Its design minimizes typical simulation drawbacks by careful abstractions specific to 6TiSCH. It makes no attempt at simulating physical behavior that can be better studied with real hardware such as: synchronization issues due to imperfect crystals, bit-specific transmission errors, hardware-dependent processing delays. Instead, it focuses on simulating the behavior that is observed in the network *from* the MAC layer. We achieve this with two abstractions. First, we quantize time in TSCH slots: an event can only take place at the slot boundary. Second, we abstract the protocol messages to only

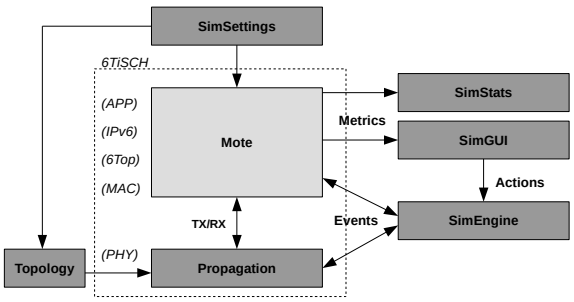



Figure 6.4: Internal architecture of the 6TiSCH Simulator.

carry semantically-relevant parameters: exchanged messages are not byte-accurate.

We build upon these two abstractions to provide a simulator that can accurately monitor:

- the behavior of the scheduling function in response to generated traffic (in frames/second).
- the behavior of the routing protocol in response to topological changes.
- the behavior of 6P in response to MAC-layer drops.
- the behavior of the application in response to scheduling, routing, and network stack configuration.

Different simulator instances can be scheduled to execute in parallel on available processor cores (tested up to 56 cores).

The simulator is implemented in approximately 8000 lines of Python code, spanning 6 core files and a graphical interface[2]. The internal architecture of the simulator is shown in Fig. 6.4. The main component is `Mote`, where the major part of the 6TiSCH stack is implemented. It is configured by `SimSettings`, which contains the input parameters introduced by the user. `Mote` also generates metrics for `SimStats`, and schedules events that are scheduled and processed by `SimEngine`. Some of these events are the TXs and RXs, which are evaluated by `Propagation` depending on the existing `Topology`.

Now that we defined our tools, we show in the following sections how we use them to estimate the performance of TSCH-based networks. In Section 6.3, we compare real-world results with simulator ones. We replay the real-world routing graph, link quality and network configurations in the simulator, and apply different scheduling algorithms. We try to find the distance

[2]In contrast, other network simulators, such as NS-3 in the current 3.28 version, have up to 1,483,072 lines when considering "*.cc*", "*.h*" and "*.py*" files.

between commercial products performances and the theoretical optimum. In Section 6.4, we generalize our approach and estimate the 6TiSCH performances under a wide range of scenarios.

## 6.3 Real-World vs. Simulation

To have a better idea of the capabilities of TSCH, I first wanted to know its performances in commercial products, and use this as a point of reference. To do so, I deployed a SmartMesh IP network in a building across three floors and recorded its network statistics, particularly looking for three Key Performance Indicators (KPIs): E2E Upstream Latency, E2E Reliability, and Network Lifetime. All three are described in detail in the following paragraphs. I then compared those reference results against simulator results using the same network configuration (e.g. data rate, topology, link quality). My goal is *not* to verify the accuracy of the simulator (this is already done in **municio18simulating**), but to quantify and understand the difference between theoretical and experimental TSCH limits.

We deploy a 50 node SmartMesh IP networks in both Inria-Paris office buildings (as it is done in the EvaLab presented in Chapter 4) for 72 hours, and log all network events and network statistics. We could not reuse network statistics we collected in Chapter 4, as we wanted to test the network in various schedule configuration. We then "replayed" the topology and links connectivity in the simulator. Finally, we compare the results against theoretical limits.

### 6.3.1 Experiment Description

We deployed 50 Analog Devices DC9025 boards equipped with external antennas on the ceilings of 4 floors in the Inria-Paris building. Each node has two parents, and sends its temperature (27 B payload) every 30 seconds.

We ran 3 experiments for 24 hours each, and logged the network statistics. In each experiment, we varied a parameter called *base bandwidth* (`base_bw`). The base bandwidth is the expected time interval between packets, in ms. We ran our experiment with the base bandwidth values 10,000, 20,000 and 30,000. The smaller the base bandwidth, the more cells are allocated. We thus expect the experiments with $base\_bw = 10000$ to have a shorter E2E US latency and a smaller network lifetime than the experiment with $base\_bw = 30000$.

### 6.3.2 Replaying the Experiment

To replicate the SmartMesh IP network, we use the same configurations: 50 nodes sending a 27 B message every 30 s.

We disable RPL and force the topology so that the network graph and links quality (PDR) are the same as in the real deployment. In SmartMesh IP, each node has two parents whereas in the default 6TiSCH configuration of RPL there is only one parent per node. In our simulation, we chose to use only the parent with the best PDR, or the root if the root is one of the two parents (to reduce the load of the one-hop nodes).

To force the topology in the simulator, we first disable RPL and its tasks (e.g. sending of DIO and DAO, rank calculation) as well as the joining mechanisms (such as the secure join). Then, based one the topology we extracted from the real deployment traces, we force the neighbor tables, the preferred parent, and the time source neighbor (used for time synchronization). We only start the simulation when all neighbors are set up. As we want to take into account the topology changes over time, the real deployment trace and the simulation topology are synchronized. When the simulation reaches the time of an event in the trace file, the simulation topology is updated (including the neighbor tables and preferred parents).

Like the topology, the scheduling task is ran every time there is a change in the trace file. An overall schedule is built and then pushed into the nodes schedule. From the nodes' point of view, the scheduling changes are instantaneous. Again, this is somewhat artificial, as in 6TiSCH nodes need to exchange data to modify their schedule.

In IEEE802.15.4-2015, there are 6 different types of timeslots:

- TxDataRxAck. A timeslot during which the node sends some data frame, and receives an acknowledgment (ACK) indicating successful reception.
- TxData. Similar to the previous, but no ACK is expected. This is typically used when the data packet is broadcast.
- RxDataTxAck. A timeslot during which the node receives some data frame, and sends back an ACK to indicate successful reception.
- RxData. Similar to the previous, but no ACK is exchanged.
- IdleListen. Time slot during which a node listens for data, but receives none.
- Sleep. Time slot during which the node's radio stays off.

We use the results from the radio chip (LTC5800-IPM) datasheet to determine the charge consumed by the mote in type of timeslot. This enables us to estimate the energy spent by each mote, at any time. The values are listed in Table 6.2.

We estimate the lifetime of each node assuming a 2821.5 mAh capacity (a pair of Energizer L-91 AA batteries, accounting for a 10% decrease due to

| Type of slot | Charge |
|---|---|
| TxDataRxAck | 54.5 $\mu C$ |
| TxData | 52.0 $\mu C$ |
| TxDataRxAckNone | 54.5 $\mu C$ |
| RxDataTxAck | 32.6 $\mu C$ |
| RxData | 28.6 $\mu C$ |
| IdleListen | 6.4 $\mu C$ |
| Sleep | 0 $\mu C$ |

Table 6.2: The charge spent by a mote for each type of timeslot.

manufacturing differences). We measure the energy spent from the moment the mode joins, until the end of the simulation. In SmartMesh IP, we measure the time delta and energy delta between the first and the last `HRDevice` (the first HR is sent 15 min after the mote joined).

### 6.3.3 Simulating the Limits

#### Cascading Load-Based Scheduling

We first calculate each mote's required Load, that is, the total number of cells each node requires. Then we sort the nodes by decreasing Load order. Starting with the most loaded node, we allocate the cells the node requires for its own traffic to be transfered to the root in a time (i.e. slot_offset) increasing manner. We then repeat this for each node, until all required cells are scheduled.

**Algorithm 1** Cascading Scheduling Algorithm

```
Calculate Load of each node
for node ∈ nodes do
    node.Load ← ∑ Tx(node) × k_np + Rx(node) × k_cn
Allocate cascading cells for each flow
slotOffset ← 0
for node ∈ nodes, ordered by decreasing Load do
    child ← node
    while child != root do
        getNumberOfRequiredTransmissions(child, child.parent)
        for each requiredTransmissions do
            allocateFirstAvailablePairOfCell(child, child.parent)
            slotOffset ← slotOffset + 1
        child ← parent
```

This algorithm matches (6.9).

To calculate the number of retransmissions required on a flow, we use $R_i = R^{1/h}$, where $R_i$ is the expected reliability for a link, $R$ the expected E2E reliability, and $h$ the number of hops in the flow (i.e. the number of links). This leads to a sub-optimal total number of retransmissions (see (6.8)). A greedy algorithm could be used to find the best number of retransmissions per hop so that the sum of the retransmissions is minimized on the path considered.

We use flow-prioritization to make sure that when using a Tx cell, if multiple frames are available in a node's frame queue, the frame associated with the cell's flow is sent fist. In 6TiSCH, this is done using the concept of *Track*. Both cells and frames can be associated with a *trackID*. We use that identifier to prioritize the frames.

When multiple frames are available, a node's frame queue and none of this frame is associated with the current cell *trackID* (or no *trackID* is associated with the current cell), the first frame to send is the one that has the oldest timestamp. This is known as the *Oldest First* mechanism. This mechanism helps reduce the maximum latency (we did not quantify the gain in our study). Note that this mechanism can be extended to work with multiple types of QoS such as presented in Kausa **gaillard16kausa** where certain frames have more stringent latency requirements than other.

### Simulation Parameters

We ran the simulator under the configurations summarized in Table 6.3.

We run the simulator for 20,000 slotframes. With an average of 600 slots per slotframe and a slot duration of 7.25 ms (the SmartMesh IP value), this corresponds to 24 h of simulated time, i.e. the same as the SmartMesh IP traces we collected.

| Parameter | Value |
|---|---|
| num. nodes | 50 |
| num. runs | 100 |
| num. channels | 15 |
| app data rate | 1 pkt every 30 s |
| slot duration | 7.25 ms |
| num. sloframes | 20,000 |
| target minimum E2E US reliability | 99.999% |
| target minimum E2E US Lifetime | none, then [1-3] years |

Table 6.3: The 6TiSCH Simulator Parameters.

**Simulation Results**

Table 6.4 lists the results obtained from the simulation. We explain how we obtained those results in the next section.

| | SmartMesh IP | | | simulation | | | |
|---|---|---|---|---|---|---|---|
| | bbw_10s | bbw_20s | bbw_30s | min | yr1 | yr2 | yr3 |
| E2E US Latency (s) | 9.76 | 22.21 | 17.28 | 4.69 | 11.06 | 17.33 | 23.37 |
| Net. Lifetime (yrs) | 2.99 | 2.99 | 2.82 | 1.08 | 1.88 | 2.63 | 2.98 |
| E2E US Reliability (%) | 100 | 100 | 100 | 100 | 100 | 100 | 100 |

Table 6.4: The 6TiSCH Simulator Results.

### 6.3.4 E2E Upstream Latency

The first result we look at is E2E upstream latency. To verify the equations presented in Section 6.1, we start by using only the first network snapshot collected from the 24 h SmartMesh IP network with $base\_bandwidth$ =10 s. By applying the Load-based Cascading scheduling algorithm (Algorithm 1), we find that the slotframe size is 415 slots and the maximal cascade size is 54 slots. Note that (6.9) also results in a slotframe size of 415 slots. From (6.10), we expect the maximum E2E upstream latency to be $(571 - 1 + 75) \times 0.00725 = 4.67625$ s.

In Fig. 6.5, we plot the latency CDF for the SmartMesh IP network in its 3 configurations, and the simulator results in its 4 configurations. In the first simulation configuration (`sim_loadbased_min`), we minimize the latency by setting the slotframe length to the smallest schedule length we found (given by (6.9)). In the following simulation configurations, we increase the slotframe length so that the network lifetime increases by 1, 2 and 3 years, respectively (given by (6.4)). The number in parenthesis in the legend indicates the maximum E2E US latency of 99.999% of the packets. We can see that the schedule that minimizes the maximum E2E upstream latency offers a shorter latency than the SmartMesh IP network in its 3 configurations, with 99.999% of the packets delivered within 4.69 s. This number is slightly below the expected maximum E2E upstream latency defined in the previous paragraph.

We can also see that lowering the `base_bw` SmartMesh IP parameter lowers the E2E US latency for 99%, but after that limit, the last percents are not following the same trend (the 99.999% value for $base\_bw = 30000$ is lower than for $base\_bw = 20000$).

Looking at the simulation results for the configurations that aim at extending the network lifetime, we can clearly see that our solution is not good at (and not designed for) reducing the average E2E US latency of the packets, but outperforms the SmartMesh IP latency when looking at the extreme values.

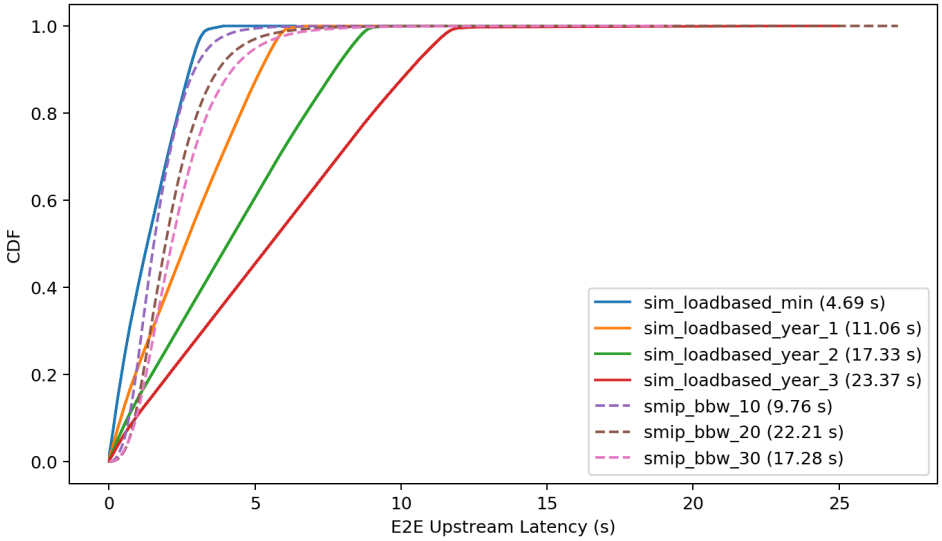



Figure 6.5: End-to-end Upstream Latency CDF. The value in parenthesis indicates the maximum value for 99.999% of the packets.

### 6.3.5 Network Lifetime

The second result we look at is network lifetime. In Fig. 6.6, we plot the lifetime CDF of each mote for the SmartMesh IP network in its 3 configurations, and the simulator results in its 4 configurations. We define network lifetime as the time until the first mote runs out of battery, i.e. the minimum of the motes' lifetimes. The number in parenthesis in the legend indicates the minimum lifetime. We can see that the schedule that minimizes the maximum E2E upstream latency also presents the worst network lifetime.

The first simulation configuration is only looking at latency and tries to minimize the maximum E2E US latency. For the 3 other configuration, we increase the slotframe size by adding Idle frames at the end of the slotframe. This will increase the network lifetime but also increase the E2E US latency. For the last 3 simulation configuration, we respectively added 388, 787 and 1184 slots to the slotframe. We obtain this value by looking at the average energy consumption of the first simulation (without extra Idle slots) and computed the number of Idle slots that needed to be added to increase the network lifetime by 1, 2 and 3 years respectively.

The network lifetime for each network configuration (simulated or real) is lower than expected. From the SmartMesh IP Performance Estimator, the expected lifetime of a 50-node network sending messages every 30 s is above 6 years. We believe that this is due to the presence of links with poor quality. If nodes cannot find at least two parents with good link quality (PDR stable and above 50%), they are forced to use unstable links. This largely increases

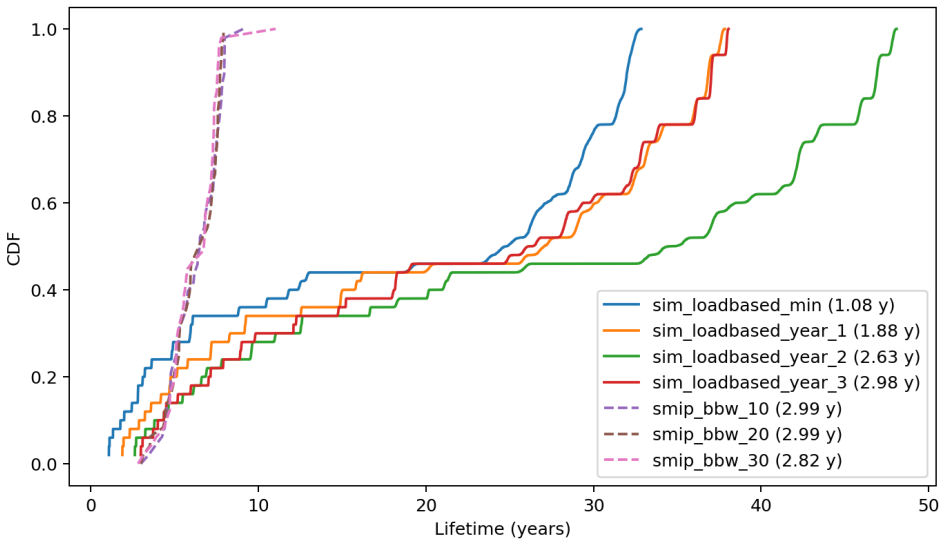



Figure 6.6: Motes' Lifetime CDF. The minimal value in parenthesis indicates the network lifetime.

the number of retransmissions and topology changes. This illustrates the fact that device placement and density are important and must not be forgotten when deploying real-world networks.

Compared to simulation, we attribute the higher network lifetime of the SmartMesh IP results to the use of two parents per mote. Using multiple parents allows to balance the load between the different available paths and thus to reduce the load of the most loaded nodes (the ones that reduce the network lifetime). In SmartMesh IP, each node uses its two parents fairly.

When the network is formed and we know that no other node will join, SmartMesh IP has the ability to turn off advertisement messages (i.e. IEEE 802.15.4 Enhanced Beacon). After turning advertisement off, no other node is able to join the network. Turning advertisement off saves energy. In the SmartMesh IP Performance Estimator, the average electric charge required for advertisements is set to 16.3 $\mu A$ for instance. This could correspond to years of network lifetime. In the network we deployed, we left the advertisement on. This further explains the low network lifetime we observe in Fig. 6.6

### 6.3.6 Discussion & Future Work

#### Virtual Frame Drop Limit

One issue with this approach is that we introduce a virtual limit on the number of retransmissions after which frames are dropped, even if the frame could be delivered within the latency requirement. Rather than dropping the frames

after a given number of retransmissions, we could keep on trying to transmit until the frame reaches a limit lifetime. Not dropping a packet after a certain number of retransmissions would however impact queue size and use schedule resources reserved for another frame. While there is space for optimization, this subject requires an in-depth evaluation.

#### Payload Size & Fragmentation

In the simulator, we do not take into account the payload size to calculate the energy consumption. We consider the worst case, that is, when frames are 127 B long. A finer energy model could be user that take into account the time need for the transmission of a frame depending on its size.

We also only studied unfragmented packets. As explained in Kausa **gaillard16kausa** the probability of successful transmission of a packet is the product of the probability of successful transmission of all its fragment. Our equation currently do not take this into account.

#### Adaptivity to Changes

In this section, we only considered applications with a fixed resources requirement. This is not always the case in reality. Applications might dynamically adapt the amount of cells they require. For time-critical applications where resources requirements might change over time, and if those requirements need to be done as fast as possible not to slow down the new traffic, we argue that they should plan ahead and reserve resources for the worst-case scenario. If new application requirements are not required as soon as possible and will not slow down the new traffic while reserving the new resources, the new schedule can be entirely rebuilt and adapted to the new traffic.

Our goal is to understand the capabilities and limits of TSCH, thus, we do not consider routing changes. Routing changes would however have a large impact on E2E latency, energy consumption and overall reliability, especially with one parent topologies. In Section 4, we show that those routing changes can be lowered to a few per days on a 23-node network. We thus assume that the topology stability should be the first objective to take into account. For the remaining, reserving flow resources over multiple parent topologies should be investigated to ensure that at least one track is always available (or at least for 99.999% of the packets).

### 6.3.7 Conclusions

In this section, we compare the results obtained from a real network deployed in a building office, with results from a copy of that same network in the 6TiSCH simulator under different scheduling configurations. This experiment confirms the equations presented in Section 6.1. Given a target mini-

mum E2E upstream reliability, we show that we are able to predict the upper performance bounds of a TSCH network taking into account the topology and links quality. There is a clear trade-off between E2E upstream latency and network lifetime that we are now able to quantify.

## 6.4 6TiSCH Performance Estimator

Testing the performances of a new lower-power wireless solution using simulation is not enough. Testing the performances of a new lower-power wireless solution using testbeds is not enough. To accurately validate a WSN protocol, the protocol has to be tested under various conditions (or at least in the worst-case conditions). To answer such demand, we propose the 6TiSCH tisch Performance Estimator, a tool that combines simulation and testbed, along with network statistics obtained from real-world deployments to provide network performance estimations when designing protocols for the IIoT.

### 6.4.1 Trace-based Simulation

The default propagation model implemented in the 6TiSCH Simulator is based on the Pister-Hack model **le09energy** This model is used to obtain the initial RSSI value of each pair of nodes in the network. This value is calculated by subtracting a uniform variance of 40 dB from the received signal strength calculated using the Friis model. The model was independently verified in an experimental setting **brun16intuitive** RSSI values are then converted to Packet Delivery Ratio (PDR) values by a conversion table that is based on real-world deployments[3]. It accurately reflects the relationship between the RSSI and PDR in large indoors industrial scenarios in the 2.4 GHz band. This model does not, however, reflects the PDR variation over time. We shown in Chapter 5 that this parameter can have a large impact on network performance.

What we need is a way to replay the connectivity traces we collected in previous work in order to precisely reflect the radio connectivity. We modified the 6TiSCH Simulator in order to do so. We replaced the default Pister-Hack propagation model and made the simulator use trace files instead. As said earlier, the time is quantized in TSCH slots and the length of a slot is known. It is therefore possible to map the simulation time to the trace file time. The simulation connectivity then follows directly the trace connectivity. If the simulation runs for longer than the traces, the simulator connectivity model starts replaying the trace from the start, i.e. it “loops”. If the simulator has more nodes than the trace, then the simulation is aborted.

[3] The connectivity traces were obtained from the `http://wsn.berkeley.edu/connectivity/` project at the University of California, Berkeley, lead by Thomas Watteyne and Prof. Kris Pister. The dataset used is “soda”, created by Jorge Ortiz and Prof. David Culler.

Using only one trace results in unrepresentative outputs. The simulation needs to runs on a series of traces in order to have significant results.

### 6.4.2 Inputs and Outputs

The input and output parameters are similar to the ones presented in the SmartMesh IP Performance Estimator we presented in Chapter 4. The user provides a topology (i.e. the number of motes per hop in a table or a file) and an application data-rate for each mote. On top of that, the user can select the Scheduling Function (SF), or the RPL routing policy. The 6TiSCH Performance Estimator then outputs a series of KPIs estimations (e.g. latency, network lifetime).

The main difference with the SmartMesh IP Performance Estimator, apart that it is based on the 6TiSCH standards, is that the 6TiSCH Performance Estimator uses per-link and time-varying link quality whereas the SmartMesh IP Performance Estimator uses a link quality that is fixed and the same for each link. The user can select a connectivity trace from a list of files (i.e. the connectivity traces we collected during this thesis).

The output KPIs are extrapolated from similar simulator results. As we do not want to run simulations every time the user requests a performance estimation, we pre-run simulations and extrapolate the results to match the user requested combination. If the user asks for 45 nodes and the simulation was run only for 40 and 50 nodes (with the same scheduling and routing), we can estimate the value for 45 nodes. Every time a new version of the simulator is released, the simulator runs a series of different combinations (e.g. number of motes, application data-rates, schedules) and produce KPIs. From those KPIs we extrapolate the estimated performances of the network that corresponds to the user configuration.

### 6.4.3 Status

The 6TiSCH Performance Estimator is still on development. We believe that this tool (or a similar one) should be used to estimate the performances of 6TiSCH networks and be part of the 6TiSCH protocol development process. To address Industrial IoT challenges and build networks that can be depended upon, we need a common way to test and validate the network performances.

## 6.5 Summary

In this chapter, we study the limits and trade-offs of TSCH-based networks. We start by presenting the theoretical limit of E2E upstream latency and how it trades off against network lifetime. The latency can be infinite due to the fact that the probability of an infinite number of consecutive retransmissions is not null. However, we show that we can provide bounds when observing

a subset of the packets. Following a similar approach, we show that network lifetime and its trade-off against latency can also be quantified. We then compare results obtained from simulation with network statistics from real-world traces, and explain the observed differences. We show that the commercial product we use has shorter E2E US latency on average but longer maximum latencies than when using an algorithm to minimize maximum the E2E US latency. Finally, we present the 6TiSCH Performance Estimator, a tool for low-power wireless protocol designers to estimate the performance of a 6TiSCH network.

# Chapter 7

# Conclusion and Perspectives

Currently, commercial IIoT products offer wire-like end-to-end reliability (>99.999%) together with very long network lifetime (>10 years). While those solutions are not fully standardized yet, they will soon be, and we consider E2E reliability issues as a solved problem. In the meantime, network solutions that address time-critical applications are at their early development stage, and their latency performance bounds are still not fully understood. This thesis contributes to filling this gap and understand the performance limits and trade-offs of TSCH-based networks in terms of e2e latency and network lifetime.

## 7.1 Contributions

To start with a clear understanding of the current IIoT challenges, we deployed 4 real-world low-power wireless sensor networks:

- **PEACH**: A frost prediction system for peach orchards in Mendoza (Argentina). We deployed 23 devices on a $11km^2$ area to measure temperature and humidity at different locations in the orchard.
- **SnowHow**: A snow-pack monitoring system in the Sierra Nevada (California). We deployed 27 devices in a $100km^2$ area to measure snow level, solar radiation and other environmental data in the mountain.
- **EvaLab**: A Smart Building monitoring system in Paris (France). We deployed 22 devices in a $800m^2$ area to measure temperature in an office building.
- **SmartMarina:** A metering and management system for marinas in Agde (France). We deployed 19 devices in a $5km^2$ area to measure the presence and electricity consumption of the boats in the marina.

We collected a total of 3M network statistics and 32M sensor data and made those publicly accessible. We believe that this is the largest real-world low-power wireless dataset available to the research community. We analyzed those datasets and found not-so-intuitive results, such as the fact that a very low number of routing changes (less than 5 per day) are actually happening in real-world deployments.

We collected radio connectivity traces on different testbeds to have data that is dense in time, space and frequency. We compared those radio connectivity traces with results obtained from the real-world deployments, and identified key metrics and behaviors that must be taken into account when developing a protocol for the IIoT. We further proposed a methodology to ensure that a protocol that performs well on a testbed also does so when moving beyond testbeds. More specifically, we looked at External Interferences, Multipath Fading, and their variations over time.

We identified the theoretical limits and trade-off points of the Time Slotted Channel Hopping (TSCH) mode of the IEEE 802.15.4 MAC layer. We focused on E2E upstream reliability, E2E upstream latency, and network lifetime. We tested those theoretical limits against real-world performances by "replaying" connectivity traces and topologies in the 6TiSCH simulator. We show that we are able to quantify the E2E upstream latency bounds in 99.999% of the network packets. We also propose a method to estimate the trade-off between network lifetime and E2E upstream latency.

Finally, we present the 6TiSCH Performance Estimator, a tool for low-power wireless protocol designers to estimate the performance of a network, given a set of inputs (e.g. network topology, connectivity traces, traffic).

## 7.2 Perspectives

### Designing Protocols for the IIoT

The available of open testbed platforms to the research community increased the impression of realism in the networking protocol validation process. Yet, a solution validated on a testbed might not work well when deployed in real-world conditions. We still lack a generic way of characterizing networks experimentation. The IoTBench initiative proposes an architecture to characterize a network experiment considering its *inputs*, *outputs*, and *observed* metrics. I believe adopting a common benchmarking architecture is the right approach to measure and compare network experiments in a standard way, in order to produce results that one can rely upon.

### Trace-based Simulation

Simulation is a great tool for rapid evaluation and performance estimation of networking protocols. In wireless networking simulators, the component

to simulate which is most tricky is often the radio propagation. Using models that are too simple often leads to unrealistic results; models that are too complex often leads to high computation and processing time. I believe Trace-Based Simulation should be used more, as it offers a good trade-off between complexity and realism.

### 6TiSCH Performance Estimator

The 6TiSCH Performance Estimator is a tool to help protocol designers estimate the performance of their protocols. To go further than just providing performance estimation, this tool should be extended to identify the performance bottlenecks and protocols weak points. We could consider using Machine-Learning techniques to identify performance bottlenecks that are not straightforward.

Both the 6TiSCH Simulator and 6TiSCH Performance Estimator are made to test the capabilities and performances of the 6TiSCH networking stack. We envision those tools to be part of the standardization process and that they should be systematically used before validating any protocol in 6TiSCH.

### Node's Mobility

So far we only considered networks where nodes are static. Our work could be extended to encompass networks with mobile nodes. As reserving resources along a path takes times and energy, I do not believe that networks where every node is mobile can offer reliability or latency guarantees. However, I think that hybrid networks can. By having a fixed backbone the network can allocate extra resources to carry traffic of potential leaf mobile nodes. Thus, the mobile nodes only have to allocate resource with their parents, the rest of the path to the root being already reserved. Such scheduling policy needs of course to be tested and validated.

### Downward Traffic & Actuation

In this thesis, I only studied upward traffic for convergecast applications. This type of traffic encompasses periodic monitoring applications and event-driven data collecting applications. We showed that TSCH-based networks can provide predictable performances, and can be used in time-critical applications. However, we did not study the performance of such networks when using downstream traffic. Providing latency performances estimation for downstream traffic would enable the use of low-power wireless for applications that require actuation in industrial applications.

### Industrial Process Control

Industrial Process Control (IPC) is a production process where tasks are automated to optimize the efficiency of the process. IPC applications require high reliability and low latency. Schindler *et al.* introduced the first characterization and implementation of a closed-loop wireless feedback control network using TSCH over a 4-hop network**schindler17implementation** To the best of my knowledge, this is the first work on IPC using completely standards-compliant and TSCH-based network. As IIoT matures, it tends toward applications that require service guarantees and predictable performances.